NTNU
Norwegian University of
Science and Technology

Service Deployment in Heterogeneous Cloud-like Environments

Anders Nordby Gullhav

MASTERKONTRAKT
- uttak av masteroppgave

1. Studentens personalia

Etternavn, fornavn **Gullhav, Anders Nordby**	Fødselsdato **07. mar 1987**
E-post **anders.gullhav@gmail.com**	Telefon **90927100**

2. Studieopplysninger

Fakultet **Fakultet for Samfunnsvitenskap og teknologiledelse**	
Institutt **Institutt for industriell økonomi og teknologiledelse**	
Studieprogram **Industriell økonomi og teknologiledelse**	Hovedprofil **Anvendt økonomi og optimering**
E-post **anders.gullhav@gmail.com**	Telefon **90927100**

3. Masteroppgave

Oppstartsdato **17. jan 2011**	Innleveringsfrist **13. jun 2011**

Oppgavens (~~foreløpige~~) tittel
Service Deployment in Heterogeneous Cloud-like Environments

Oppgavetekst/Problembeskrivelse
Purpose:
The purpose of this work is to develop better tools for deployment decision problems for actors providing heterogeneous, replicated services in a cloud-like environment. Validation of the tools depends on realistic parameter values for e.g. cost, capacities, structures, usage patterns.

Department of Telematics (ITEM) – NTNU will assist in making the necessary assumption with respect to technology and parameter values while testing the tools.

Main contents:
1. Describe the deployment planning problem.
2. Formulate one or more models for the planning problem.
3. Implement the one or more sub model(s) for the problem by use of appropriate software.
4. Test the model(s) with real data.
5. Discuss the applicability and adequacy of the model(s) as decision support tools for the given problem.

Hovedveileder ved institutt **Professor Nygreen Bjørn**	Biveileder(e) ved institutt **Associate Professor Poul Heegaard (ITEM)**

Merknader
1 uke ekstra p.g.a påske.

4. Underskrift

Student: Jeg erklærer herved at jeg har satt meg inn i gjeldende bestemmelser for mastergradsstudiet og at jeg oppfyller kravene for adgang til å påbegynne oppgaven, herunder eventuelle praksiskrav.
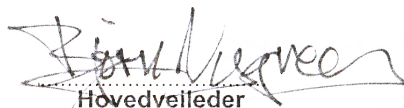
Partene er gjort kjent med avtalens vilkår, samt kapitlene i studiehåndboken om generelle regler og aktuell studieplan for masterstudiet.

NTNU 14/1 - 2011

Sted og dato

_____
Student

_____
Hovedveileder

Originalen oppbevares på fakultetet. Kopi av avtalen sendes til instituttet og studenten.

I have travelled the length and breadth of this country and talked with the best people,

and I can assure you that data processings is a fad that won't last out the year

Abstract

The amount of power consumed by data centres world wide is increasing, and due to growing electricity bills, service providers aim more attention on energy-efficient management of their data centres. In order to achieve this goal, a service provider need to make smart decisions regarding the deployment of his services. At the same time, in order to satisfy his end-users, a service provider needs to focus on delivery of services complying with the quality of service (QoS) requirements. Consequently, he needs to make decisions related to replication level of his services, as well.

In this thesis, I propose two interrelated mixed integer linear programming (MILP) models aiming at supporting service providers in their decision making. The first MILP concerns energy-efficient deployment of a service provider's services in his own virtualized data center, where the objective is to minimized the cost of energy usage, while satisfying the response time and availability requirements of the end-users. The second MILP introduces the flexibility of Cloud computing by letting the service provider have the opportunity to deploy services in a public cloud, and hence the objective is to minimize the total cost of deployment, while still, ensuring satisfactory QoS levels.

The proposed MILP models are tested on test instances of varying size with the intention to discuss scalability issues and commenting on modelling choices. The results show that the second model is the hardest to solve, in terms of closing the optimality gap, but nevertheless, it is depicted that good solutions are found early in the branch and bound search. Furthermore, different modelling choices illustrate the trade-off between energy-efficient management of data center resources and service performance.

Preface

This Master's thesis is the result of the final work done in order to accomplish a Master of Science degree with specialization in Managerial Economics and Operations Research at the Department of Industrial Economics and Technology Management at the Norwegian University of Science and Technology (NTNU).

This work has combined concepts from computer science, dependability and performance analysis of ICT systems and operations research, which have made me utilizing much of the knowledge I have acquired during my five years of study at NTNU. In this respect, I feel that the work with this master's thesis have been inspiring and challenging, and thus I look forward to continue my work in further studies.

I would like to thank my supervisor, Professor Bjørn Nygreen, for discussions, his advices and all the time he has devoted me. Moreover, I thank my co-supervisor, Professor Poul Heegaard, at the Department of Telematics for his contributions in technical issues and for proposing this thesis in the first place. Although, not as involved as in the last semester, I feel that I still owe a thanks to graduated PhD Mate Csorba for giving me inspiration to proceed with this topic. Lastly, I also owe thanks to my girlfriend for letting me spend long nights writing on my thesis, without being too angry.

Note to the reader: Some paragraphs in Section 2 and Sections 4.4.1 to 4.4.3 is based on the background section in my specialization project (Gullhav, 2010).

Contents

List of Figures

List of Tables

1 Introduction

Providers of cloud software services are faced with several decisions regarding the deployment of their software components. The end-users require require low response time and downtime on the services they use, while the service providers also have to deal with utilization and power management of their hardware resources. These facts lead to the need for an analytical approach for service providers to make decisions about their service deployment.

The relationship between the service provider and his end-users' is handled through a Service Level Agreement (SLA), which is a formal contract including a specification of the services to be delivered and the quality requirements of the delivered services. The quality of service (QoS) requirements, specified in an SLA, often include both performance and dependability requirements, and to satisfy these, the service provider has to implement some means. Fault tolerance, through replication of software components, is often used to ensure service dependability. Replication might also be used to ensure the appropriate response time of a service, in that a number of copies of the same software component serves the demand from the end-users' at the same time. Thus the service provider needs to make decisions regarding the replication level of the software components and where to deploy the replicas.

The energy usage of data centres has increased considerably over the past few year. Brown et al. (2008) report that the estimated energy consumption of US data centres in 2006 was 61 billion kWh, or about 1.5 percent of the total US electricity consumption. Furthermore, they state that the energy consumption was more than doubled from 2000 to 2006 and that it is expected to double again by 2011. Beloglazov et al. (2011) argue that the increasing energy consumption of computing systems has started to limit further performance growth, and hence the objectives in computer system design have shifted from performance improvements to power and energy efficiency. This means that simultaneously as the service provider has to ensure that the delivered services satisfies the end-users' QoS requirements, he must focus on energy-efficient management of his data center(s).

A strategy used to minimize the power consumptions in a virtualized data center is to consolidate the virtual machines, running the services, on a minimum set of physical computing nodes, in order to turn off some of them. However, due to variability in service demand, this might lead to a situation where some virtual machines might not get the required amount of resources, which lead to decreased performance and higher response times. Hence, cloud service providers faces a trade-off between power efficiency and performance (Beloglazov et al., 2011).

With the advent of the cloud computing paradigm, the deployment decisions of a service provider does not only involve decisions about on which physical nodes in his own data center the software components should be deployed. In fact, Cloud computing gives the service provider the opportunity to deploy his services in other, public data centres, denoted public clouds, as well. Cloud computing is an internet technology allowing for on-demand resource scaling, aiming at increasing the utilization of computing resources. Several firms are already investing large amounts of money in data centres, where service providers and other companies can lease hardware and network resource. Amazon is, through Amazon Elastic Compute Cloud (EC2), providing a web service where users can acquire computing capacity on demand (Amazon EC2 website, 2011). Of course, there are still security and trust issues involved when deploying software in a public cloud, and a lot of research is done regarding these concerns, but, despite the security issues, the usage of public clouds are becoming more and more popular. The attractiveness of Cloud computing lies in the on-demand resource scaling properties, and this is especially attractive for service providers that are providing a service with varying demand. Such service providers need to install enough hardware to make their service handle demand peaks, which would imply that a lot of hardware resources are idle at times with low demand. Besides, the usage-based pricing model of Cloud computing may be tempting. Thus when considering the possibilities that Cloud computing bring forward, a service provider needs to determine whether or not to lease resources in a public cloud, and if he decides to do so, he must make a decision about which software components that should be deployed in his own data center and in the public cloud.

Up to now, I have describes some of the tactical decisions a provider of cloud software service will need to make. In a longer time horizon a service provider has to make strategic decisions related to investment in data center infrastructure and, of course, decisions related to his service portfolio. Taking into account the cloud computing paradigm, a service provider might decide to scale down his investment in hardware, and instead spend more money on leasing hardware resources from public cloud providers.

However, the focus in this thesis will be on the tactical and operational decisions faced by a service provider, and in this regard, the decisions related to the replication level of the services and the deployment of the resulting replicated services components. Two mixed integer linear programming (MILP) models are developed in order to support the decision making, where the first MILP only considers deployment in the service providers virtualized data center, whereas the second takes into account the opportunities to deploy services in a public cloud, as well. The replication level decisions are based on both principles from queueing theory and reliability block diagrams, and in order to deliver a service corresponding to the SLA requirements of the end-users, the virtual machines, running the replicated service components, are assigned a dedicated amount of resources on the physical infrastructure. The objective in the first MILP model is to minimize the power consumption, while ensuring an appropriate service quality. As the second MILP model introduce possibilities for deployment in a public cloud, the total cost of deployment, including cost of energy usage in the service provider's data center(s) and cost of using public cloud infrastructure, is sought minimized.

The goal in this work is then to construct MILP models which can be used to support service providers in their decision making, both regarding the replication level and the deployment architecture of the resulting replicas, and thereafter test the models in order to discuss their applicability and adequacy. This thesis will also challenge the trade-off between power consumption and performance, through a set of submodels.

The outline of the thesis is as follows: Next, in Section 2, I will give an overview of the concepts of Cloud computing. Section 3 describes the drivers of power consumption in data centres and presents a relationship be-

tween the power usage and the utilization of the servers' central processing unit (CPU). Thereafter, Section 4 details the concepts of service performance and dependability and give an introduction to analytical models for quantifying the response time and availability of ICT systems. These first sections aim to present the necessary background and methodology in order to understand the proposed mathematical programming models. Section 5 gives an overview of related work on Service Deployment Problems (SDP), mainly focusing on approaches regarding performance and energy efficiency in distributed systems, like Clouds. Suddenly, in Section 6 the service deployment problems considered in this thesis are presented together with the MILP models, while Section 7 considers the implementation of the models in commercial software and presents and discusses the numerical results obtained. Lastly, Section 8 concludes my work and Section 9 draw lines of future work.

2   Overview of the Cloud computing concepts

Cloud computing is the most recent computing paradigm which promises to deliver computing as the fifth utility, after water, electricity, gas and telephony, over the Internet. As the cloud computing model still is an evolving technology, its definition is also still evolving. The current definition of Cloud computing by the National Institute of Standards and Technology (NIST) is: *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, 2009). 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 idea of cloud computing is then to make the network of resources, as well as the management of the resources, hidden from the service provider or end-user, thereby letting the resource pool be viewed as a cloud. It is often assumed that Cloud computing is continuing the earlier paradigm shift from mainframes to client-server models. However, Voas and Zhang (2009) points out that the cloud computing paradigm might also seem to return to the original mainframe paradigm, where in the latter, simple terminals were connected to powerful mainframes shared by many users. Likewise, comparing the PCs of today with the powerful Internet cloud, the PCs seem like lightweight terminals.

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

2.1 Service models in Cloud computing

NIST defines three service models for Cloud computing (Mell and Grance, 2009): Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS). In the SaaS service model cloud-users are delivered software applications over the Internet. This eliminate the need for the cloud-users to install and run applications on their own computers, facilitating management and maintenance. Examples of such a service model are Google Docs, Salesforce.com and webmails like Gmail and Hotmail. In the IaaS service model the cloud-user is provided fundamental computing resources like processing power, storage and network, onto which the cloud-user can deploy and run own software, including operating systems and applications. The cloud-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 Enterprise Compute Cloud (EC2) (Amazon EC2 website, 2011) and Ubuntu Enterprise Cloud (Wardley et al., 2009). 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, onto which the cloud-user can deploy own applications. Examples of PaaS are Google AppEngine and Force.com.

2.2 Cloud deployment models

Furthermore, when considering cloud deployment models, NIST distinguishes between four different types of clouds (Mell and Grance, 2009). 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, that are bound together by some sort of technology that enables data and application portability. That is, the VMs have to be packaged in a standardized format to allow for dynamic movement between clouds. The Amazon Machine Image format (Amazon EC2 website, 2011) can be used for this purpose as it is supported by other cloud solutions as well, e.g the Ubuntu Enterprise Cloud. In such a way, a service provider running a private cloud based on the Ubuntu Enterprise Cloud would be able to dynamically deploy VMs, according to the demand for his services, into the public Amazon EC2 cloud. The last MILP model presented in this thesis is based on a hybrid cloud environment.

2.3   Cloud security

Considering the security concerns arising with usage of public clouds, Armbrust et al. (2010) points out that the users face security threats both from outside and inside in the Cloud. Many of the outside threats are similar to those already threatening large data centres. In an IaaS cloud, the cloud user is responsible for application-level security, while the cloud provider is responsible for the physical security. The authors emphasize that the responsibilities of the cloud user may be outsourced, to a third party who sell speciality security services, with ease because of the homogeneity and standardized interfaces of platforms like Amazon's EC2. In such a manner, Cloud computing might make the handling of security threats from the outside easier. On the other hand, Cloud computing introduce the new problem of threats from the inside, where the cloud providers must guard against attacks from the users, and thus protect the users from each other. Another aspect of cloud security is the legal matters that may follow. Kaufman (2009) ask questions related to the legal concerns emerging when personal information is stored in the clouds, for example; who has jurisdiction over data as it flows across borders, and can governments access that information as it changes jurisdiction? She states that these questions can not be answered by the usage of current laws, and that resolving these concerns will take years if the past decade of attempts of making the current laws fit into the internet world is used as a standard.

2.4 Cost models in public clouds

Many public cloud service providers (e.g. Amazon) have adopted at pricing scheme that let the customers pay for only the actual usage of their applications. For many businesses and organizations this pay-as-you-go model may seem attractive since the businesses do not need to invest in infrastructure, but instead utilize the economies of scale of the cloud providers and lease computing resources. Thereby, they are turning capital expenses to operating expenses. There are also other economic benefits of Cloud computing, as stated by Armbrust et al. (2010). They argue that elasticity and transference of risk, especially risk of underprovisioning and risk of overprovisioning, are important economic benefits of Cloud computing. Furthermore, they state that this transference of risk also outweighs eventual loss occurred if the pay-as-you-go pricing was more expensive than buying and depreciating a comparable infrastructure (e.g. server) over the same time period. The cost structure of the pay-as-you-go model in public clouds vary among the different cloud IaaS providers, but they have some similarities listed below[1]:

- Deployment costs, i.e. the costs of running a VM instance per time unit,

- Communication costs, i.e. costs per gigabyte transferred in and out of an instance,

Besides, some public cloud IaaS providers charges extra cost for hourly usage of load-balancers and extended monitoring services. In addition, this on-demand pricing scheme, Amazon AWS is offering long term contracts to its customers, where VM instances are reserved for one or three years against a one-time payment. This scheme leads to lower hourly costs for the cloud-users. In the models in this thesis, the focused pricing scheme in the public clouds is the on-demand model, and furthermore the models only take into account the hourly costs of deploying VM instances and not the communication costs. The latter cost component is left for future work.

[1]The cost structure presented here is based on (Amazon EC2 website, 2011), (Rackspace Cloud hosting website, 2011) and (GoGrid Cloud hosting website, 2011)

3   Power consumption in data centers

As already stated, the data centres world wide currently consume a notice-able amount of the total electricity consumption, and hence a potential area of cost reduction for data center managers and service providers is by im-plementing energy-efficient management of their infrastructure resources. The focused infrastructure of data centres in this thesis is the physical servers constituting the data centres, and hence I will in this section give an introduction to the current drivers of power consumption in data center servers and a relationship between power consumption of servers and the CPU utilization.

Beloglazov et al. (2011) give an overview of the sources of power con-sumption in a data center and a survey of the research done in the area of energy-efficient management of data center resources. Minas and Ellison (2009) in Beloglazov et al. (2011) present data indicating that main part of power consumption in servers is the CPU, followed by the power supply unit (PSU) efficiency loss and the memory. The distribution of power con-sumption between the components of server is shown in Figure 1. In the figure it is seen that the CPU is not dominating the total power consump-tion, as was the case in earlier servers. Beloglazov et al. (2011) state that this is due to the improvement of the CPU power efficiency, and especially the advent of low-power modes. As a result of this, current CPUs consume only 30 percent of their peak power consumption at idle times. Although the idle power consumption of CPUs is decreasing, other components in the server are still power inefficient in idle state, and thus an idle server might still consume more than 70 percent of the peak power (Beloglazov et al., 2011).

Fan et al. (2007) presents a relationship between the CPU utilization and the total power consumption by a server, and they show that this relationship can be expressed linearly as in (1).

$$p(u) = P_{idle} + (P_{busy} - P_{idle})u \tag{1}$$

$p(u)$ represents the total power consumption as a function of the CPU uti-lization, u, P_{idle} refers to the power usage when the server is idle, while
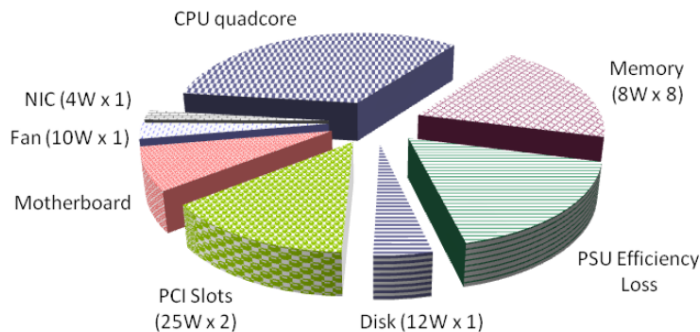
Figure 1: Power consumption by the components of a server (Minas and Ellison, 2009 in Beloglazov et al., 2011)

P_{busy} is the power usage of the server at peak load. Figure 2 shows the measurements done in (Fan et al., 2007), together with the linear model in (1) and an empirical non-linear model that more closely fits the measurements. The empirical non-linear model uses a calibration parameter r which minimizes the squared error. The authors report that r is set to 1.4. Beloglazov et al. (2011) state that extensive experiments have shown that the models predict the power consumption with an error below 5 percent for the linear model and below 1 percent for the empirical model. In the mathematical programming models presented in Section 6, the linear relationship in (1) is applied in order to quantify the power consumption of the servers in a data center.

Beloglazov et al. (2011) argue that the main reason for power inefficiency in data centres is due to the low utilization of the servers, but also acknowledge that there is a trade-off between power efficiency and service performance. This is because modern service applications cannot be run on fully utilized servers as small fluctuations in demand may then lead to performance degradation. Barroso and Hölzle (2007) report two key observations related to the utilization of data center servers. They state that servers are rarely kept idle or fully utilized. Instead servers operate often between 10 percent and 50 percent of their maximum capacity.
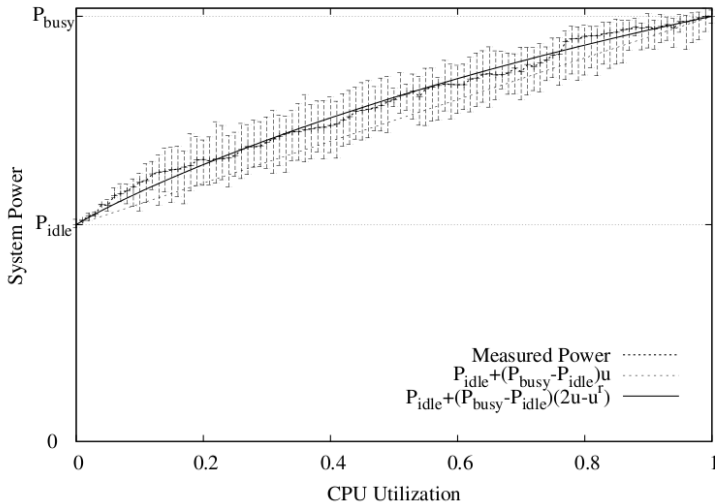
Figure 2: The measured relationship between total power consumption of a server and the CPU utilization (with error bars) compared with a linear model and an empirical model ($r = 1$ of the relationship (Fan et al., 2007)

Moreover, Beloglazov et al. (2011) distinguish between two techniques to dynamic power management on the hardware level in data centres. The first technique lets servers be turned off or enter a low-power state in periods of inactivity, in order to eliminate the power consumption of having idle servers turned on, cf. equation (1). Benini et al. (2000) state that this technique comes with a drawback, in that the transitions from a low-power state to the active state might cause delays and additional power usage. Another idea, instead of turning servers off, is to decrease or increase the clock frequency and the supply voltage according to the demand. This idea is adopted in the Dynamic Voltage and Frequency Scaling (DVFS) technique. The models in this thesis will apply the former of these two techniques, but as the models only considers a single time period, switching costs of turning servers on and off are not regarded.

4 Service performance and dependability

This section will give an overview of the service performance and dependability concepts used throughout this thesis. But firstly, I will give a definition of the service notion used throughout this thesis, and a brief introduction to service level agreements, the formal contract between a service provider and its end-user.

4.1 Service definition

A service in this thesis is defined as an aggregation of different subsystems, denoted as *components*. These components are software modules, collaborating in order to deliver the specified service to the end-users. The partitioning of services into components can be done on different levels. In this thesis the components reflect the various types of servers that is needed for service delivery, e.g. web servers, database servers, application servers, authentication servers or authorization servers. I assume that the different components of a service not necessary are deployed on the same physical server infrastructure, and thus I treat the services as distributed services. A typical architecture of a distributed service is the *three-tier model*, where one distinguishes between the presentation tier, the logic tier and the data tier. The presentation tier handles interactions with the end-users directly, while the logic tier evaluates the clients requests and processes data between the presentation tier and the data tier. The information is stored in the data tier (e.g. database server). Web applications is often modelled using this three-tier model, where the presentation tier represents the webserver the client is directly connecting to. Other architectures of distributed services includes *n-tier models* and *peer-to-peer models*. Figure 3 shows an example of a simple web service modelled using the three-tier model.

In addition to the distinction between the service notion and the component notion, the components might be replicated in order to increase the performance and dependability of the service. This means that a component can be viewed as being composed of a set of replicas. Replication of components will be explained in more detail in sections 4.3.3 and 4.4.3.
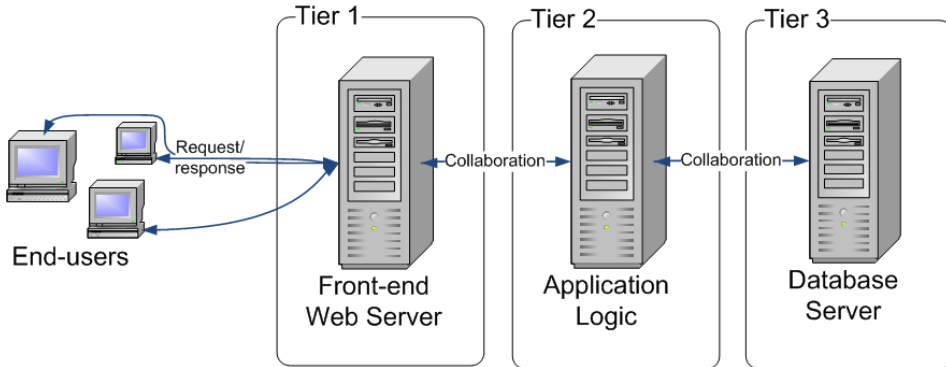
Figure 3: Example of a *three-tier model* that consist of a web-server (tier 1), an application logic (tier 2) and a database (tier 3).

4.2   Service level agreements

A service level agreement (SLA) is a negotiated contract between the service provider and the end-users of the service. Emstad et al. (2008) states that an SLA may include:

- a description of the service or the set of services that is to be delivered to the end-user,

- the quality of service (QoS) parameters of the service, and how they are measured and the tolerance levels they are to be kept within,

- the amount of traffic the user can transmit, and

- the consequences the service provider faces if the QoS parameters exceed the tolerance levels

In their description of SLAs and QoS, Emstad et al. (2008) define QoS as *the degree of compliance of a service to the agreement that exists between the user and the provider of this service.* There exist several interpretations of the QoS notion, and ITU-T (1994) defines it as *the degree of satisfaction*

of a user of the service. This definition is more related to the experience of the end-user, and the definition of Emstad et al. (2008) may fit better in an engineering context where a system is to be dimensioned to meet the requirements of the end-users. I will therefore use the first of these two definitions of QoS in this thesis. Furthermore, Emstad et al. (2008) describes a QoS parameter as a random variable characterizing the service, and such variables include both performance and dependability metrics.

In the next two subsections I will elaborate on the concepts of service performance and dependability, and touch the techniques used in this thesis to quantify the performance and dependability of a service. For now, the two concepts will be treated separately, although they are tightly coupled. In Section 6, they are joined in order to make proper decisions on the dimensioning of the services.

4.3   The performance concept

Performance, in the context of ICT-systems, is defined as *the ability of a system to provide the resources needed to deliver its services*(Emstad et al., 2008). Hence, if a service provider is to provide a service with high performance to its end-user, the service components need to have access to an amount of resources that corresponds to the demand. Now, I will present the focused performance metric, and next, give a brief description of an analytical model used to quantify this metric.

4.3.1   Performance metrics

There exist several metrics for quantifying the performance of a service, including *throughput*, the portion of system capacity that is utilized by the users, and various system times. The system times comprise the *waiting time*, the time a service request is pending for service in the system; the *service time*, the time a service request is served by the system; and the sum of the two preceding, the *sojourn time*, the total time a service request stays in the system (Emstad et al., 2008). Menasce and Almeida (2001) define the *response time* of service as the sum of the sojourn time and the

network latency, including protocol overhead and transmission delay. In the models presented in this thesis, I focus on the response time metric rather than the throughput, and furthermore, I do not model the network latency. Thus I treat response time and sojourn time as the same quantity.

4.3.2   Response time calculation

The focused analytical models for calculation of the response time in this work is queuing models, based on birth-death processes. I will not go deep into the queuing theory concepts, but rather touch the concepts used in the replication level decisions in the service deployment models, presented in Section 6. There are written several books about queueing theory, and this section is based on the works of Emstad et al. (2008) and Zukerman (2010).

From the definition of a service, we have that a service is composed of several collaborating components. Hence I choose to view a service as a queueing network. For simplification, I assume that the queueing network is acyclic, that is, a service request is not being processed in the same component more than one time. Simultaneously, I assume that the the arrival process is Poisson, the service time is negative exponentially distributed and there is infinite buffer space, such that the network is made of Markovian queues without loss. When this is the case, Burke's theorem yields that the departure process of a queue, in steady state, is also Poisson, and hence, the arrival rate equal the departure rate.

In order to calculate the response time, or sojourn time, of a service, it is first necessary to calculate the response time of each component. Using Kendall's notation, if the component is treated as an M/M/1 queue, with a given arrival rate λ and an expected service time of $1/\mu$, the expected response time, $E[T]$ is given as

$$E[T] = \frac{1}{\mu(1-\rho)} = \frac{1}{\mu - \lambda} \tag{2}$$

where $\rho = \lambda/\mu$ is referred to as the utilization. For an M/M/1 queue the utilization must take a value in the interval $[0, 1)$ to be considered stable.

Response time as a function of utilization
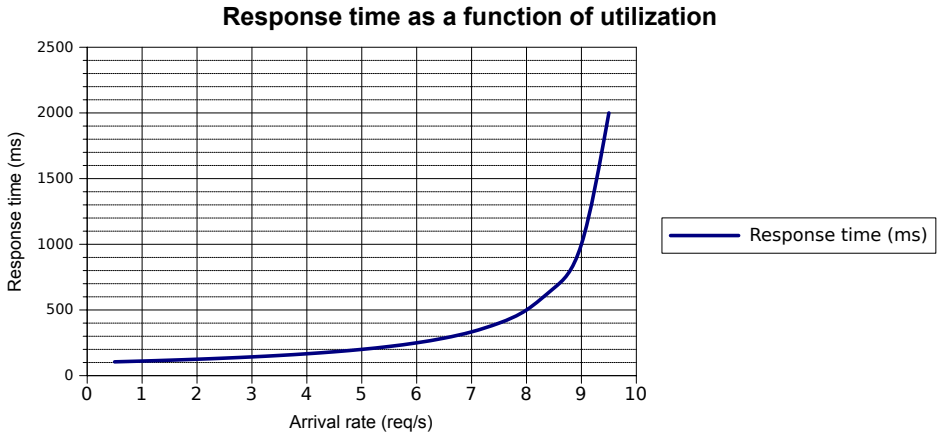
Figure 4: The response time curve of an M/M/1 queue for values of λ between 0 and 9.5 request per second and $\mu = 10$ requests/sec.

Figure 4 shows the relationship between the response time, in an M/M/1 queue, and the arrival rate, λ, given an excepted service time. In the figure, the values of λ range from 0 to 9.5 requests per second and the service rate μ equals 10 requests per second. As illustrated, the response time increases dramatical when the utilization passes a certain threshold, and this threshold is referred to as the "knee of the curve".

If the component is treated as an M/M/k queue, with a given arrival rate λ and an expected service time of $1/\mu$, the expected response time, $E[T]$ is given as

$$E[T] = \frac{1}{\mu(k - \rho)}(P(W > 0) + k - \rho) \tag{3}$$

where $P(W > 0)$ is the probability of waiting, which for the loss-less systems considered here is given by Erlang's C-formula. The stability condition of an M/M/k queue is given by $\rho < k$.

The calculation of the total time a request spends in an acyclic queueing network, i.e. the service response time, can be done using Burke's theorem

and (2) and (3). In fact, the expected response time of a service is the sum of the expected response time in each component. Given that a service is composed of a set, $\mathcal{Q}$ of components, viewed as $|\mathcal{Q}|$ M/M/1 queues, and the request arrival rate into each component, $q \in \mathcal{Q}$, is λ_q and the expected service time of a request in each component is μ_q, then the expected service response time is

$$E[T_{SERV}] = \sum_{q \in \mathcal{Q}} E[T_q] = \sum_{q \in \mathcal{Q}} \frac{1}{\mu_q - \lambda_q} \tag{4}$$

4.3.3   Replication to increase performance

Out of equations (2) - (4) and Figure 4 we can see that it is important to keep the ratio between λ and μ low in order to obtain a good response time. This means that the components need to be assigned or have access to an amount of resources (eg. CPU power) that corresponds to the demand, i.e. keep the utilization below the knee of the curve. One way of keeping the utilization on a tolerable level, is by letting several identical components serving the demand. These identical components, all serving demand, are often termed active replicas, as opposed to passive replicas, which will be elaborated in the next section. In such a case a load-balancer can be used distribute the requests to the active replicas in a round-robin or totally random fashion. This approach will lower the utilization of each replicated component, and hence increase the performance through better response times.

4.4   The dependability concept

Dependability is defined as *the trustworthiness of a system such that reliance can justifiably be placed on the service it delivers* (Laprie et al., 1992). The goal of this part is to point out which types of means that can be used in order to achieve the required degree of trustworthiness. To be able to do this, one must first understand what is causing a system to not being trustworthy and how trustworthiness can be quantified.

4.4.1   The dependability tree

The causes of an untrustworthy system is grouped into a notion termed threats, and the quantification of trustworthiness is done through a set of dependability attributes. The relationship between this threats, attributes and means can be illustrated by a tree, commonly known as the dependability tree. The dependability tree is shown in Figure 5. The next paragraphs will give an overview of the relevant concepts of threats, attributes and means.
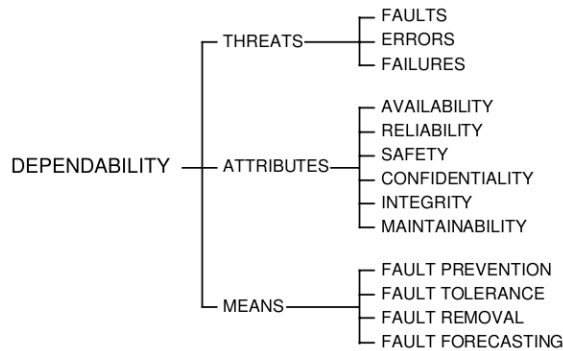
Figure 5: The dependability tree, illustrating the connection between the dependability concepts: threats, attributes and means (Avizienis et al., 2001)

Threats

The threats to dependability are defined as failures, errors and faults. Failures are defined as *the transition from correct service delivery to incorrect service delivery.* An error is defined as *the part of a system state which is liable to lead to failure.* Lastly, a fault is defined as *the adjudged cause of an error*(Laprie et al., 1992). Faults, errors and failures can be expressed as a sequence of events. Faults may lead to errors in the internal state of the system. This error can in turn lead to a failure, i.e. an event that makes the system deliver an incorrect service. Furthermore, failure in one (sub)system can be the cause of a fault in another (sub)system. There

are several types of faults that can incur in a system, and Avizienis et al. (2004) groups them into development faults (eg. software bugs), physical faults, i.e. faults affecting hardware, and interaction faults, which include all faults that have been propagated into the system by interaction with another faulty system.

Attributes

To quantify dependability there is defined different attributes that describe the properties of a system. Two of these attributes are availability and reliability. The reliability of a system is defined as *its ability to provide uninterrupted service*, and it is quantified by, for example, the mean time to first failure. On the other hand, the availability of a system is *its ability to provide a set of (correct) services at a given instant of time*. Measures of availability are, for example, the mean uptime of a system and the asymptotic availability (Helvik, 2007). The chosen dependability attribute in the subsequent models is availability, quantified using the asymptotic availability, i.e. the probability that a system is working at a randomly chosen time in the future. An analytical model for quantifying the availability of a service is given in section 4.4.4.

Means

When developing a dependable system there are four techniques, or means, that can be utilized. These means are:

- **Fault prevention:** how to prevent the occurrence or introduction of faults

- **Fault tolerance:** how to provide a service complying with the specification in spite of faults

- **Fault removal:** how to reduce the number and severity of faults

- **Fault forecasting:** how to estimate the present number, the future incidence, and the likely consequences of faults

Fault prevention and fault removal is closely related and can be grouped into a concept named fault avoidance, i.e. how to aim for fault-free systems (Avizienis et al., 2004). Both fault avoidance and fault tolerance techniques may be used to meet the dependability requirements of a system, and the use of one of them does not exclude the other. A balanced usage of both approaches is necessary (Helvik, 2007).

Fault tolerance is the only of this means that will be used explicitly in the modelling of distributed services in the MILP models in section, but the other means are included here to bring a simple overview over the dependability concepts. A deeper description of fault tolerance will be given in the next subsection.

4.4.2   Fault tolerance

In fault tolerance, the basic principle is that a fault is allowed to produce an error, but the error is not allowed to cause a failure. Hence, the aim of fault tolerance is to avoid failures in spite of faults. To prevent errors from causing a failure it is necessary to detect the errors and either removing them or compensate for the errors by introducing enough redundancy to enable the delivery of an error-free service from the erroneous state. The design of a fault tolerant system is dependent on which classes of faults that the system should tolerate (Avizienis et al., 2004).

Gärtner (1999) has done two observations regarding the fault tolerance of distributed systems. The first observation yields that no matter how well designed or how fault tolerant a system is, there is always a possibility of a failure if faults are too frequent or too severe. The second observation is that to be able to tolerate faults, one must employ a form of redundancy. Helvik (2007) defines redundancy as the addition of resources in the form of hardware, software, information or time beyond what is needed for normal provision of system services. The service models in this thesis will only explicitly employ redundancy in terms of additional subsystems, i.e. replication of service components.

4.4.3   Replication for fault tolerance

There exists different approaches to redundancy of software systems, and there is distinguished between modular redundancy and standby redundancy (Helvik, 2007). Only the latter one is regarded here. In order to explain a the standby redundancy approach, consider a service consisting of one component that is replicated through three active replicas in order to fulfill the response time requirement (cf. Section 4.3.3). Let us assume that the service complies with the response time requirement if and only if three or more replicas are active at the same time. In such a case, if a replica fails and the response time raises to an intolerable level, the end-user will not consider the service as available. This lead to the need for some extra replicas that could take over the service delivery in case of a failure in one of the active replicas. I refer to these replicas as passive replicas, since they remain passive until a failure occurs.

Standby redundancy is divided into different classes according to the degree of updated state information in the passive replicas, compared to the active replicas. One distinguishes between cold or lukewarm standbys, where the passive copies have no or little state information; and hot standbys, where the passive copies have an updated or almost updated version of the state space of the active. Hot standbys will require a higher communication frequency than cold standbys in order to keep the state updated. Letting the passive replicas having highly updated state information is advantageous for services that require short interruption in service delivery, as higher updated state often means shorter failover delay, i.e. the time when a failure occurs until a passive replica is ready for service delivery. The replicated service components modelled in this thesis is assumed employing hot standby redundancy.

When applying a redundancy approach it is important to ensure that the deployment architecture of the replicated components does not lead to a situation where replicas fail dependently of each other, i.e. a failure of one replica should not imply that another replica fails. Hence, if a standby redundancy scheme is used to make the service components able to deliver correct service despite potential hardware faults, then two replicas of

the same software component should not be placed on the same physical hardware. This deployment constraint is referred to as node-disjoint deployment. Domain-disjoint deployment is another term, which state that replicas of the same component is to be deployed in different clusters, or domains, of physical machines. Such a deployment constraint is a means to increase the availability related to link failures and failures in the management systems controlling a cluster.

4.4.4   Availability calculation of replicated services

The purpose of this subsection is to show how replication affect the availability of a service. Hence, in the subsequent availability calculations, the failures related to the physical machines, the communication links or any other systems, other than the replicated service components themselves, are not taken into account.

Regarding the definition of a service presented earlier, we have that a service consists of a multiple of components. In order to simplify the calculations of the availability of a service, I assume that the components fail independently of each other, and that the restoration time of a component is independent of whether other components have failed. Furthermore, I assume that the asymptotic availability of a component is known. When these assumptions are taken it is possible to compute the availability of a service by using the asymptotic availability of each component constituting the service. This approach is called *reliability block diagrams*. Helvik (2007) gives a nice presentation of this approach and the theory below are based on his book.

The reliability block diagram approach uses the structure of a system, here service, as a basis for the analysis. In my case, each of the components of a service is called a reliability block, and together they form a series structure. Figure 6 shows a simple service with three components in a series structure.

A service modelled as a series structure fails if one of its components fail, and hence, the asymptotic availability of the service is given by the product of the asymptotic availabilities of each component. Using the example of a
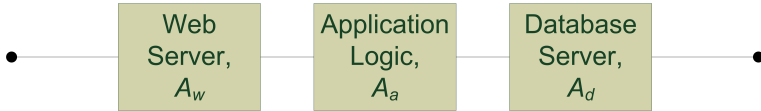
Figure 6: Reliability block diagram: series structure of service composed of a web server, an application logic and a database server, with asymptotic availability A_w, A_a and A_d, respectively.

3-tier service in figure 6, the availability of the total system is given by

$$A_{SERV} = A_w A_a A_d \tag{5}$$

where A_w, A_a and A_d is the availability of the web server, application logic and database server, respectively. Generalizing the formula for a service consisting of a set of components, $\mathcal{Q}$, where component q has availability A_q, gives the following:

$$A_{SERV} = \prod_{q \in \mathcal{Q}} A_q \tag{6}$$

Since the asymptotic availability is the probability that a system is working in a randomly chosen time in the future, the availability of each component takes a value between 0 and 1. Therefore, the total availability of a series system is upper-bounded by the availability of the component with the lowest availability.

As stated above, I introduce redundancy in order to make the system fault tolerant and increase the availability. Figure 7 shows a reliability block diagram of the same service as in Figure 6, but now employing redundancy in each component. For the sake of adding another dimension to the calculations, I say that the web server need to have two active replicas in order to serve the requests from the end-users with a tolerable response time, while the other two components only have one active replica each. In order to increase the availability of the total service, the web server and the application logic has one passive replica each, whereas the database server has two passive replicas, in case of failures.
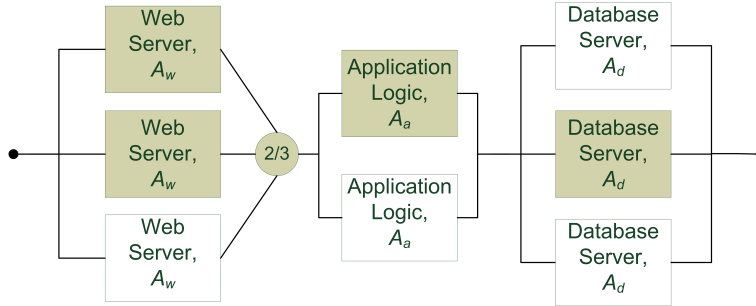
Figure 7: Reliability block diagram: complex structure of service composed of replicated web servers, replicated application logics and replicated database servers, with asymptotic availability A_w, A_a and A_d, respectively. The shaded blocks represent active replicas, while the non-shaded represent passive replicas

A component having one active and a number of passive replicas is said to form a parallel structure, and in Figure 7 the application logic and the database server form parallel structures. A component having k active and $n - k$ passive replicas $(k \leq n)$ is termed a *k-out-of-n* structure, and the web server in Figure 7 is said to have a 2-out-of-3 structure. A k-out-of-n structure is in fact a generalization of a series and parallel structure, since setting $k = n$ gives a series structure and setting $k = 1$ and $n > k$ gives a parallel structure. In order to calculate the availability of the service in Figure 7, we firstly calculate the availability of each component and then use (6) to calculate the total availability. Computation of the availability of a component, q, forming a k-out-of-n structure, hence also series and parallel structure, is done as follows:

$$A_q^{k-out-of-n} = \sum_{j=k}^{n} \binom{n}{j} A_q^j (1 - A_q)^{n-j} \tag{7}$$

(6) and (7) gives the following formula for calculating the total asymptotic availability of a service:

$$A_{SERV} = \prod_{q \in \mathcal{Q}} \sum_{j=k_q}^{n_q} \binom{n_q}{j} A_q^j (1 - A_q)^{n_q - j} \tag{8}$$

where n_q is the total number of replicas of component q and k_q is the number of active replicas of q. Formula (8) together with (4) can be used in the decision making related to the number of active and passive replicas of the service components, and these decisions will be elaborated in Section 6.

5 Related work on service deployment problems

This section will present some work done in the area of service deployment, scheduling or resource optimization and approaches to power management in data centres, and focuses mainly on approaches done with exact solution methods. Most of the reviewed work takes into account performance aspects, in different manners, while a few only considers energy-efficient management of data centres.

Xiong and Perros (2006) proposes an approach for resource optimization in the computing environment of a service provider. The problem to be solved consists of minimizing the cost of computer resources allocated to high- and low-priority customers, while satisfying the customers' SLA requirements. The authors emphasize that using a statistical bound of the response time as an SLA metric is more meaningful for the customers than using the mean response time, which is typically used in literature. The authors use queueing theory for quantifying the response time and derive Laplace-Stieljes transforms (LST) of the response time distribution for high- and low-priority customers, and they present two algorithms for solving the resource optimization problems.

Hermenier et al. (2009) present solution procedures for dynamically allocating tasks (VMs) to nodes in clusters and creating reconfiguration plans for migration of the VMs, using constraint programming. The first problem is referred to as the Virtual Machine Packing Problem (VMPP), and has the objective of minimizing the number of nodes used, in order to same energy, while ensuring that each VM has access to sufficient memory and CPU power. The resulting problem formulation is a multiple knapsack problem, and it is solved using dynamic programming. The second problem, denoted the Virtual Machine Replacement Problem (VMRP), creates a reconfiguration plan for each possible configuration using the number of nodes determined by the VMPP and chooses the one with the lowest migration costs, i.e. the cost of moving a VM from a node to another.

Van den Bossche et al. (2010) introduce a BIP model that is used to deploy a set of applications, in a hybrid cloud environment, while minimizing the total costs of the service provider for executing the applications.

The cost of executing an application consists of data traffic costs and the costs of resource consumption in the public clouds, and each application consists of different tasks with deadline constraints. The private clouds are capacitated in terms of CPU power and memory, and hence tasks must be outsourced if there is not free capacity in the private clouds, such that the deadline is achieved. It shows that runtime of the BIP in the hybrid cloud environment increases dramatically when the number of applications are increased, and the authors propose development of custom heuristics in future research.

Speitkamp and Bichler (2010) present an algorithm on a problem denoted the Static Server Allocation Problem (SSAP), which basically consists of finding mappings between services and capacitated physical machines, and minimize the cost of the physical machines needed. The authors state that the SSAP is strongly NP-hard, even when only one type of resource is considered, and all servers have the same cost and capacity. They also extend the problem to include time variable workload and constraints, including the maximum number of services to deploy on a server and constraints stating that a subset of the services have to be deployed on different physical servers. On problems with identical physical servers, the authors propose to reduce the number of equivalent solutions by differentiate the cost of physical servers by a small amount. For the extended SSAP they propose an heuristic algorithm, consisting of two phases, where the first phase solves the LP-relaxation of an original MIP formulation and the second phase find an integral assignment of those services that were fractionally assigned in the first phase. This heuristic is compared with branch-and-bound, and additionally, the author compute a lower bound on the number of servers, in order to compare the solution time and solution quality. The authors report that he algorithm allowed them to solve large problem sizes within minutes, with good solution quality.

Petrucci et al. (2010) have taken an approach to dynamically manage the cluster power consumption through a MIP 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 minimize power consump-

tion while meeting the performance requirements of the applications. The problem is modelled as a variant of the one dimensional variable sized bin packing problem, which according to the authors, is known to be NP-hard. The authors also extend the problem to take into account switching costs for turning servers on and off and migration costs for changing the application deployment configuration from one time period to another. The model is solved by the use of CPLEX 11, and the authors set a solution tolerance gap to 5 percent with respect to the optimal solution, in order to allow the solver to provide acceptable solutions in short amount of time. The authors report that by setting this optimality gap criteria the approach scales well for clusters up with 350 servers.

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 center level and power control is conducted on the individual servers, by turning servers on and off and using Dynamic Voltage and Frequency Scaling (DVFS). The objective in the model is to minimize the total electricity cost for data centres in a multi-electricity-market environment. The decisions in the model includes the number of servers turned on and the CPU frequency in each data center and the amount of traffic from a set of web portal servers to the data centres. They model delay constraints at each data center by treating a data center as an M/M/n queue, where n is the number of turned-on servers, and also have a constraint on the maximum CPU frequency at each data center. The proposed model is a mixed integer non-linear program (MINLP) and the authors use the Generalized Benders Decomposition (GBD) for solving the model.

The research presented in this section considers mostly load-balancing and frequency scaling of the physical servers as a means to achieve good performance. On the other hand, my work uses replication level decisions in combination with deployment decisions to achieve the same. I have not discovered any research considering these decisions jointly, and thus this thesis might present a novel feature to deployment problems.

6 Models of the service deployment problems

This section presents two main interrelated Service Deployment Problems (SDP), which differ with respect to the environment where the services are to be deployed. Now, there will be given brief description of the two problems, and then, the underlying assumptions related to the two main parts of the problem, namely the services and the data centres or clouds, are specified. Thereafter, some concerns about the replication model is elaborated, and lastly the two problems are formulated as mixed integer linear programming (MILP) models.

6.1 The service deployment problems

A deployment, in the context of this work, is mapping between the logical software components, composing a service (cf. Section 4.1), and the physical machines in a data center. The software components are assumed packaged in virtual machines (VMs) before being deployed onto the physical machines, and thus several VMs can be consolidated on the same physical server. The deployment environment in the first SDP consists of the physical machines in a single virtualized data center, managed by the service provider himself. In the second SDP the deployment environment is extended to include other data centres, available to the general public, i.e. public clouds, as well, giving the service provider more flexibility in the deployment decisions. In this case, the data center of a service provider is referred to as the private cloud, and thus the service provider's private cloud forms a hybrid cloud together with the public clouds of IaaS providers.

According to the SLAs negotiated with his end-users, the service provider is responsible for providing a service that corresponds with the response time and availability requirements agreed on. In order to so, the service provider need to make decisions regarding the replication level of his services, more specifically, he needs to decide the number of active and passive replicas of each component constituting a given service. Each of the resulting replicas is, as mentioned above, packaged in VMs before being deployed onto the physical machines, and there is a one-to-one relationship between

VMs and replicas. As a reason of this the terms replica and VM are used interchangeably throughout this thesis.

As noticed in Section 3, the applied power management technique focus on turning of unused nodes, and thus the model includes decisions related to this, as well. The decisions related to deployment, replication levels and turned-off nodes need to be taken simultaneously, as all decisions are affecting the cost of deployment. The cost of deployment are evaluated through a cost function, and the cost function of the first MILP model concentrates on the energy usage in the data center of the service provider, and hence promotes an energy-efficient solution. Furthermore, since the deployment environment in the second MILP model includes public clouds, the costs of deploying VMs in those clouds also have to be accounted for, resulting in a new cost function.

6.2 Model framework and assumptions

6.2.1 Service model

The end-users of a service are assumed to generate demand according to a Poisson distribution where the average number of requests per second is known. I assume that a request from an end-user is processed by the different components of the service in sequence, such that the components can be viewed as a chain of servers and that the request must processed in every server before the response is sent back to the end-user. This is in general not true, but this is done to lower the complexity in the performance evaluation of the services. Moreover, each request for a service consumes an average amount of CPU power. The amount of CPU power needed to handle a service request may vary by the type of component, and is therefore component-specific. This means that the total demand in CPU power for a component is known. On the other hand, I have chosen to not model the demand for memory or storage resources in the models herein, but implementing this is left for future work. Thus this implies that the only demanded resource in the system is processing power delivered by the CPUs on the nodes.

As stated in Sections 4.3.3 and 4.4.3, components might be replicated into a number of active and passive replicas, where the active replicas serve the demand generated by the end-users. In this relation, I assume that the demand for a given component is equally split between the active replicas of this component. Furthermore, both active and passive replicas consume an extra amount of CPU power for processing other task that is not directly related to the serving of demand. Such tasks include state synchronization between replicas and other management-related functions.

The average service time of a request being processed in a VM holding an active replica is known, and the services times are assumed to be negative exponentially distributed. Corresponding with the average service time, each VM running an active replica is assigned, or reserved, a fixed, dedicated amount of CPU power on the physical server where it is deployed. Naturally, as the demand and service time are not deterministic, the total amount of CPU power assigned to the active replicas of a component must be greater than the CPU power demanded by the service requests of the component, otherwise the queues in the VMs would be saturated. Thus, with a known demand distribution and a service time distribution it is possible to calculate the average sojourn time, or response time, of a request, using the formulas provided in Section 4.3.2.

My definition of a service implicitly promotes the need for some amount of communication between the components of a service. The models presented in this thesis neither model this communication as a cost nor as a bandwidth requirement. Additionally, in reality, the group of replicas of a component communicates internally in order to maintain an updated state, but this type of communication is not modelled either. Modelling the communication between collaborating components and internally in a group of replicas of a component is left for future models.

The availability of the service components is given as the asymptotic availability, that is the availability when the component is in steady state. It is assumed that the all replicas of a component have the same availability, and that the replicas of a component fails independently of each other. In fact, I assume that a replica fails independently of every other replica in the system. Besides, I consider the restoration time of a failed replica of being

independent of the state of other replicas, i.e. there is enough repairmen to handle a situation where all replicas are failed. These assumptions about no interdependency of replica failures and restoration times are not completely realistic, but they significantly simplify the availability computations.

Lastly, even though the second SDP problem allows for deployment of VMs in several data centres, there might exists bindings on some components, stating that the replicas of that component should be deployed in a specific data center. Such a constraint might, for example, be related to security concerns, in that a database server, containing private or confidential information, is only allowed to be deployed in the private cloud.

6.2.2   Data center models

In the first SDP, I only consider a single virtualized private data center as a possible site of deployment for the VMs running the replicated service components. In this case I assume that the service provider has full control over his own data center, and by this I mean that he can decide which node that is used for deployment of a given replica. The nodes are in the following models assumed to be homogeneous, meaning that the execution of a given replica leads to the same usage of processing power, independently of the node which runs the replica. The nodes in the private data center is modelled with capacity constraints in terms of CPU power, and since the nodes are assumed homogeneous, all nodes have the same capacity. Furthermore, since the communication between service components are not considered, neither the communication links in the data center nor the network latency is modelled in the MILP models. Moreover, the nodes are not assumed totally free of failures, but the node failures are not taken into account in the computations of the services' availability. This is done for simplification. Instead, I forbid two replicas of the same component to be deployed on the same node.

The second SDP problem adds another dimension to the deployment problem, by including the option for the service provider to deploy his replicated components in the cloud data centres of one or several public IaaS providers. In this case, the service provider does not have control

over which nodes in the public cloud that runs his services, but he might
have the opportunity to make decisions about which out of possibly many
geographically distributed data centres, herein denoted domains, to run
his replicas. Hence, the problem of the service provider becomes to find an
optimal deployment configuration, where replicas can be deployed internally
on his own managed nodes or in the public cloud. Figure 8 illustrates a
hybrid cloud environment where the private cloud is connected to a public
cloud, consisting of several geographically distributed domains.
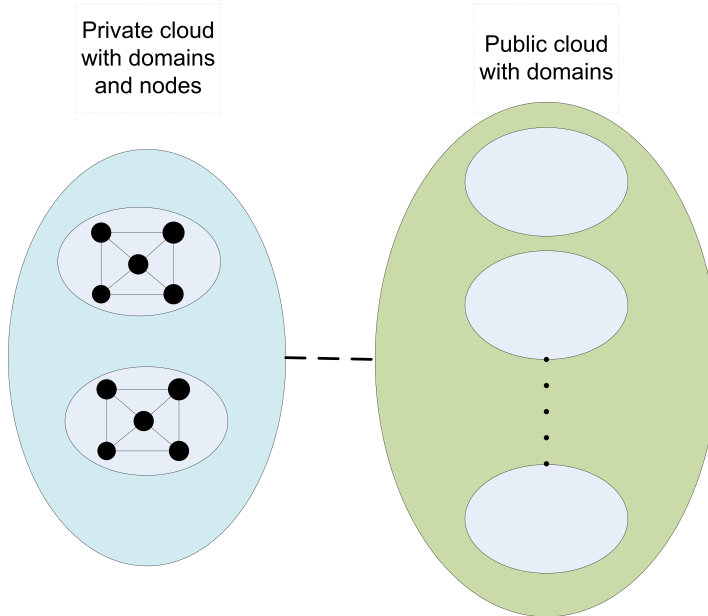
Figure 8: A hybrid cloud environment consisting of a private cloud and public cloud.
The domains are depicted by the smaller ellipses, while the physical nodes in the domains
are depicted by black circles. In contrast to the nodes in the private domains, the nodes
are not visible in the public domains.

Deployment of service components in the public clouds is also done
through the use of VM instances. However, different from the deployment

in the private data center, the VM instances provided by public cloud IaaS providers come with capacity limits, related to CPU. In order to provide flexibility to their users, public cloud IaaS providers deliver various types of VM instances, differentiated by the capacities and cost. Although, there exist, in reality, possibilities for buying large enough VM instances to run more than one replica, I prohibit such decisions in my models.

Still, since the communication between the service components is not regarded, neither the network latency in the public clouds nor the communication costs are modelled. This means that the network latency, including the inter-domain latency, is not taken into account in my response time model. Especially the latter latency component might be of importance in services with low-latency requirements, but modelling network latency are left for future work.

Likewise the nodes in the private data center model, the clouds, both private and public, are not assumed free of failures, but these failures are not accounted for in the availability computations either. In order to increase the availability related to failures of clouds, some components may have requirements on the number of clouds used for deployment of its replicas.

6.2.3   CPU power assignment to passive replicas

In order to make the time from an active replica fails to a passive replica is ready to serve demand as short as possible, one must ensure that there is a low probability of having to migrate the passive replica to another node before it can turn active. A migration might be necessary if there is not enough CPU power on the node, where the passive replica resides, in order to let the replica serve demand. There exist several solutions to overcome this, and I have modelled three different approaches. Firstly, one might assign each passive replica the same amount of resources as the active ones, meaning that all replicas could serve demand at the same time. I call this approach the *all-active approach* since all replicas are active and serving demand. Secondly, a more resource-friendly approach could be taken, which only assigns enough CPU power to serve demand to a fraction of the passive replicas, and hence I call this fraction of passive replicas as

semi-active and this approach the *semi-active approach.* These semi-active replicas is in the further models not assumed of serving demand unless a failure in one of the active replicas occurs. Lastly, it would also be possible to lower the assignable capacity on each node, such that at least one of the passive replicas deployed on a node can turn active. This last approach is denoted the *capacity-lowering approach.* Regarding the second SDP model, I use the same line of thought when modelling the deployment in the public cloud domains. Except from the fact that the service provider is not able to lower the capacity in the data center of an IaaS provider.

6.3 The Service Deployment Problem in a Private Data Center

This subsection presents and solves a problem which will referred to as the Service Deployment Problem in a Private Data Center (SDP-PDC). I have built a MILP model of the problem and implemented the formulation in commercial software to be able to test the formulation on different test instances. Comments on the implementation and the numerical results are given in Section 7.

Notational conventions in the MILP models

To be able to keep the MILP models presented in this section easy to read, I use the convention where parameters are written with a uppercase letter, the variables are denoted by a lowercase letter and the sets are denoted by a uppercase letter in calligraphic font. The indices are written with subscript letters in lowercase. In addition to this, the parameters and some variables and sets, are written with a number of extra subscript uppercase letters. Lastly, all constraints are depicted by having all variables on the left-hand side of the sign.

6.3.1 Problem definition

The problem is composed of three main decision problems, namely decisions related to the replication level of each of the service components,

the decisions related to the deployment of these resulting replicas and the decisions related to whether a node should be turned off or not. The resulting replication level and deployment decisions, together with the decisions regarding the nodes to turn off, are evaluated using a cost function promoting energy-efficient solutions. Now, the three decision problems will be described in detail, before proceeding to the mathematical formulation.

Replication level decisions

A set of services, $\mathcal{S}$, consists of a set of components, $\mathcal{Q}_i$ ($i \in \mathcal{S}$) which should be replicated in order to meet the response time and availability requirements. The response time and availability requirements are given by the parameters R_{EQRTi} and R_{EQAVi} for each service $i \in \mathcal{S}$, respectively. Hence these parameters put performance and dependability constraints on an entire service, and by the usage of equations (4) and (8), we are able to formulate the following constraints:

$$\sum_{q \in \mathcal{Q}_i} \frac{1}{\dfrac{C_{PUASSiq}}{J_{OBLOADiq}} - \dfrac{D_{EMi}}{a_{iq}}} \leq R_{EQRTi}, \quad \forall i \in \mathcal{S} \tag{9}$$

$$\prod_{q \in \mathcal{Q}_i} \sum_{j=a_{iq}}^{n_{iq}} \binom{n_{iq}}{j} A_{VAILCiq}^{j} (1 - A_{VAILCiq})^{n_{iq}-j} \geq R_{EQAVi}, \quad \forall i \in \mathcal{S} \tag{10}$$

In (9) D_{EMi} denotes the average request arrival rate for service i, assumed equal for all components of i, a_{iq} is the number of active replicas of service-component pair (i, q). I assume that the requests arriving a component is equally split between its active replicas, such that the arrival rate into a replica of pair (i, q) equals $\frac{D_{EMi}}{a_{iq}}$. Moreover, $C_{PUASSiq}$, denotes the CPU power assigned or reserved to a VM holding an active replica of pair (i, q) for handling the demand, and lastly, $J_{OBLOADiq}$ is referred to as the average amount of CPU power needed to process a request for pair (i, q). This means that I assume that average service time of a request, $\frac{1}{\mu}$, in a given service-component pair (i, q) equal $\frac{J_{OBLOADiq}}{C_{PUASSiq}}$. In real systems, it is not only the CPU power allocated to an application that affects

the service times, but also the frequency of I/O operation and memory access will have an impact. For simplicity, this is not taken into account in the models. Furthermore, in (10), $A_{VAILCiq}$ represents the asymptotic availability of a pair (i, q), and n_{iq} denotes the total number of replicas for (i, q). The variables in (9) and (10) are the number of active replicas of (i, q), a_{iq}, and the total number of replicas of (i, q), n_{iq}, and hence both constraints are non-linear. To be able to model the replication level decisions in a MILP model, I have chosen to construct an algorithm that produces a number of *replication patterns* for each service. A replication pattern for a service i defines a combination of active and passive replicas for each component of i, and the set of such patterns for service i, satisfying the response time and availability requirements, is represented by $\mathcal{R}_i$. Thus, the decision is to choose one and only one replication pattern for each service. The binary variables y_{ir} are used to state whether replication pattern $r \in \mathcal{R}_i$ is adopted for service i. Generally, the parameters denoting the number of active replicas and the total number of replicas of pair (i, q), given a replication pattern, r, are represented by A_{CTRiqr} and N_{Riqr}, respectively. Specifically, for the all-active approach the A_{CTRiqr} parameters are not needed as all N_{Riqr} replicas are assumed active. Besides, for the semi-active approach, the number of semi-active replicas to deploy is given by the ceiling of the fraction between the number of passive replicas and active replicas multiplied with a control parameter, $S_{ACOEFFiq}$. For the chosen replication pattern, r, the number of semi-active replicas of pair (i, q) is then $\lceil S_{ACOEFFiq} \frac{N_{Riqr} - A_{CTRiqr}}{A_{CTRiqr}} \rceil$, which is a trade-off between letting all passive replicas being assigned the same amount of CPU power as the active ones and letting no passive replicas being assigned more CPU power than necessary for management-related tasks. As long as a component needs passive replicas for availability purposes, this approach will make at least one of the passive replicas semi-active.

The algorithm that produces the replication patterns is the same for all three approaches and is described in Section 6.5, after the presentation of the MILP model formulations.

Deployment decisions

Generally, the active and passive replicas, resulting from a chosen replication pattern, are to be mapped to a set of nodes, $\mathcal{N}$, in the private data center of the service provider, such that the replicas of a component are deployed node-disjoint. The nodes in the data center are equal in terms of CPU capacity, and this capacity is denoted by C_{APCPU}. The VMs running the active replicas of a given service-component pair (i, q) is assigned an amount of CPU power, $C_{PUASSiq}V_{ARi}$, on the node where it is deployed, in order to serve demand. The parameters V_{ARi} have the purpose of ensuring that the response time requirements hold even with an increase in service demand and take a value greater than one, based on the burstiness of the demand. This can be compared to the approach by Xiong and Perros (2006), where they applied the statistical bound of the response time as the performance metric, instead of the mean value. Furthermore, as described in section 6.2.1, both active and passive replicas consume an additional amount of CPU power, denoted C_{PUOHiq}, which also must be taken into account in the assignments.

Specifically, when taking into account the different approaches to assign CPU power to passive replicas (cf. Section 6.2.3), all replicas in the all-active approach are assigned assigned CPU power according to $C_{PUASSiq}V_{ARi}$, while this is also true for the semi-active replicas in the semi-active approach. The assignment of CPU power to passive replicas in the capacity-lowering approach corresponds to the paragraph above.

Summing up, one must make decisions regarding on which node to deploy a replica, and according to which of the three approaches used, decide which of the replicas that should be active, i.e. serving demand, and semi-active, in the semi-active approach, while taking into account the capacity constraints on the nodes and that the replicas of the same pair (i, q) are deployed node-disjoint.

Turning off nodes

As described in Section 3, a server consumes a high amount of electrical power even if it is completely idle. Therefore, in order to save energy, I allow a server to be turned down if it is not used for deployment, and assume that a server in a off-state is not consuming power.

6.3.2 MILP formulation of the SDP-PDC

Sets

The sets explained in the problem definition is summarized below.

$\mathcal{S}$	Set of all services, indexed by i
$\mathcal{Q}_i$	Set of components of service i, indexed by q
$\mathcal{R}_i$	Set of replication patterns for service i, indexed by r
$\mathcal{N}$	Set of nodes, indexed by n

Parameters

The parameters used in the formulation are shown below, and most of them have already received attention. However, the parameters related to the power consumption are given a short review. Firstly, regarding the relationship between power consumption and utilization in (1), the power usage of an idle node is denoted P_{WRIDLE}, while the coefficient in front of the utilization (u) is simplified to $P_{WRCOEFF}$. Neither P_{WRIDLE} nor $P_{WRCOEFF}$ is indexed by node as the nodes are assumed equal in terms of power usage and CPU capacity. Moreover, the cost of energy usage is given by C_{OSTPWR}, and in order to translate the power consumption into energy usage, one need to take into account the length of the time period accounted for. The length of the time period is denoted by H_{RS}. Besides, regarding the parameters denoting CPU usage and CPU assignment, these are given as a percentage of the total CPU capacity of the nodes, which means that the CPU capacity of node is assumed to be 100 (percent). Lastly, note that the parameters A_{CTRiqr} are not used in the all-active approach, while $S_{ACOEFFiq}$ are only used in the semi-active approach.

H_{RS}	The length of the time period, in hours
C_{OSTPWR}	Cost per unit of energy used
$P_{WRCOEFF}$	Coefficient translating CPU utilization to power usage
P_{WRIDLE}	The power consumption of an idle node
D_{EMi}	The avg. number of request for service i per time unit
$J_{OBLOADiq}$	The avg. CPU power needed to handle a request in (i, q) per time unit
$C_{PUDEMiq}$	The CPU power demanded by service requests for (i, q) $= D_{EMi} J_{OBLOADiq}$
$C_{PUASSiq}$	The amount of CPU power assigned to an active replica of (i, q)
V_{ARi}	The parameter ensuring that a service can handle the peaks in demand
C_{PUOHiq}	CPU power overhead for management-related tasks in replicas
C_{APCPU}	CPU capacity on node
N_{Riqr}	The total number of replicas of (i, q) using replication pattern r
A_{CTRiqr}	The number of active replicas of (i, q) using replication pattern r
$S_{ACOEFFiq}$	Coefficient used to control the number of semi-active replicas of (i, q)

Variables

The binary deployment variables x_{iqn} indicates whether a replica of the service-component pair (i, q) is deployed on node n or not. Moreover, the binary variables w_{iqn} indicate whether a replica deployed on a node is active, or not, and v_{iqn} indicates the same for semi-active replicas. Hence, the v_{iqn} is only used in the semi-active approach. The binary variables y_{ir} have already been mentioned and indicate the choice of replication pattern for a service. As the model allows nodes that are idle to be turned off, I use o_n to indicate this. Lastly, the variables m_n are used in the capacity-lowering

approach and states the amount of CPU power to be subtracted from the CPU capacity of a node, with the intention of letting any passive replica on the node to turn active. The variables are summarized below.

$$x_{iqn} = \begin{cases} 1 & \text{if the a replica of } (i,q) \text{ is deployed on node } n \\ 0 & \text{otherwise} \end{cases}$$

$$w_{iqn} = \begin{cases} 1 & \text{if a replica of } (i,q) \text{ deployed on node } n \text{ is active} \\ 0 & \text{otherwise} \end{cases}$$

$$v_{iqn} = \begin{cases} 1 & \text{if a replica of } (i,q) \text{ deployed on node } n \text{ is semi-active} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ir} = \begin{cases} 1 & \text{if replication pattern } r \text{ is used for service } i \\ 0 & \text{otherwise} \end{cases}$$

$$o_n = \begin{cases} 1 & \text{if node } n \text{ is turned on} \\ 0 & \text{otherwise} \end{cases}$$

m_n The amount of non-assignable CPU power on node n in the capacity-lowering approach

Objective function

As the objective in this model is to minimize the costs of energy usage, and I prefer a linear model, the objective function is based on the linear relationship between power consumption and CPU utilization given in equation (1). Following the line of thought in section 6.2.1, I distinguish between the CPU power assigned to a VM holding an active replica and the CPU power needed to serve demand. Hence, when I calculate the utilization on a node, referred as u in (1), I take into account the CPU power needed to serve demand. Note that the demand is split equally between the active replicas of a component. In addition, the CPU power overhead, needed for management related tasks, is accounted for. Since the CPU power parameters are given as a percentage value of the total capacity, the utilization of a node is the sum of all used CPU power on the node. The utilization of a node becomes:

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \left(C_{PUOHiq} x_{iqn} + C_{PUDEMiq} \sum_{r \in \mathcal{R}_i} \frac{w_{iqn}}{A_{CTRiqr}} y_{ir} \right) \qquad (11)$$

where the first term caters for the CPU power overhead used by all replicas and the second term caters for the CPU power usage of serving demand on the node. Notice that one and only one replication pattern should be used for service i, and hence only one y_{ir} equals one. This equation is based on the semi-active and capacity-lowering approach, but as we will be seen soon, the resulting objective function is the same for all approaches. Moreover, putting (11) into the linear relationship between power consumption and utilization and moving the summation in front of each term, we get:

$$
\begin{aligned}
p_n =\, & P_{WRIDLE} o_n \\
& + P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUOHiq} x_{iqn} \\
& + P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUDEMiq} \sum_{r \in \mathcal{R}_i} \frac{w_{iqn}}{A_{CTRiqr}} y_{ir} \qquad (12)
\end{aligned}
$$

where p_n denotes the power usage on a node. The total power usage of the private data center is the sum of the power usage on each node, which leads to the following:

$$
\begin{aligned}
\sum_{n \in \mathcal{N}} p_n =\, & \sum_{n \in \mathcal{N}} P_{WRIDLE} o_n \\
& + P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{n \in \mathcal{N}} C_{PUOHiq} x_{iqn} \\
& + P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUDEMiq} \sum_{n \in \mathcal{N}} w_{iqn} \sum_{r \in \mathcal{R}_i} \frac{1}{A_{CTRiqr}} y_{ir} \qquad (13)
\end{aligned}
$$

The last term in (13) can be simplified due to the fact that the sum of all active replicas over n equals the number of active replicas, A_{CTRiqr}, given the chosen replication pattern r. Mathematically this relationship can be written $\sum_{n \in \mathcal{N}} w_{iqn} = \sum_{r \in \mathcal{R}_i} A_{CTRiqr} y_{ir}$, which gives the following:

$$\sum_{n \in \mathcal{N}} p_n = \sum_{n \in \mathcal{N}} P_{WRIDLE} o_n$$

$$+ P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{n \in \mathcal{N}} C_{PUOHiq} x_{iqn}$$

$$+ P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUDEMiq} \underbrace{\sum_{r \in \mathcal{R}_i} A_{CTRiqr} y_{ir} \sum_{r \in \mathcal{R}_i} \frac{1}{A_{CTRiqr}} y_{ir}}_{=1}$$

$$(14)$$

We see that the last part of the last term in (14) equals one, and hence can be ignored. The derivation from equation (11) through (14) leads to an objective function as stated in (15), where the power consumption is first translated into energy usage and then into a monetary value.

$$\min z_{PDC} = C_{OSTPWR} H_{RS} \Big(\sum_{n \in \mathcal{N}} P_{WRIDLE} o_n$$

$$+ P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{n \in \mathcal{N}} C_{PUOHiq} x_{iqn}$$

$$+ P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUDEMiq} \Big) \qquad (15)$$

As mentioned the derivation (11) - (15) is based on the semi-active and capacity-lowering approach, but is also valid for the all-active approach, where all N_{Riqr} replicas are active.

Constraints

Now, I will present the constraints that are used in the semi-active approach. Thereafter I will describe the adjustments which must be done in order to make them fit into the all-active approach and the capacity-lowering approach.

Firstly, one must ensure that each service apply one and only one replication pattern, and this is done through (16). Remember that each replication pattern for a service simultaneously satisfies the response time and availability requirements, and thus explicitly modelling these are not necessary.

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1, \quad \forall i \in \mathcal{S} \tag{16}$$

Next, it is necessary to ensure that all replicas are deployed on the nodes and that the right number of replicas are active and semi-active. The sets of constraints (17) - (19), complemented with (20), guarantee the preceding. (20) provides that a pair (i, q) can have an active or semi-active replica on a node if there is a replica deployed there, i.e. $x_{iqn} = 1$. Note that (17) together with the binary definition of x_{iqn} implicitly leads to node-disjoint deployment of the replicas of a component.

$$\sum_{n \in \mathcal{N}} x_{iqn} - \sum_{r \in \mathcal{R}_i} N_{Riqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{17}$$

$$\sum_{n \in \mathcal{N}} w_{iqn} - \sum_{r \in \mathcal{R}_i} A_{CTRiqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{18}$$

$$\sum_{n \in \mathcal{N}} v_{iqn} - \sum_{r \in \mathcal{R}_i} \left[S_{ACOEFFiq} \frac{N_{Riqr} - A_{CTRiqr}}{A_{CTRiqr}} \right] y_{ir} = 0, \ \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{19}$$

$$w_{iqn} + v_{iqn} - x_{iqn} \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{20}$$

As mentioned, the nodes are capacitated, and thus a set of constraints, (21), is needed in order to not overload the nodes with replicas. Note that both active and semi-active replicas is assigned the amount of CPU power needed for service provisioning, and that all replicas are assigned additional CPU power, C_{PUOHiq}, for processing tasks other than the requests from the end-users. These constraints also prevents a node from being turned

off ($o_n = 0$) when there are replicas deployed on it.

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUASSiq} VAR_i w_{iqn} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUASSiq} VAR_i v_{iqn} +$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUOHiq} x_{iqn} - C_{APCPU} o_n \leq 0, \quad \forall n \in \mathcal{N} \tag{21}$$

Lastly (22) - (26) define all variables as binary.

$$x_{iqn} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{22}$$

$$w_{iqn} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{23}$$

$$v_{iqn} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{24}$$

$$y_{ir} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, r \in \mathcal{R}_i \tag{25}$$

$$o_n \in \{0, 1\}, \quad \forall n \in \mathcal{N} \tag{26}$$

In order to adjust the formulation above to the all-active approach, there is necessary to change and remove a couple of constraints, whereas the objective function remains the same. Firstly, the w_{iqn} and v_{iqn} variables can be removed from the formulation as all replicas are considered active, and hence the constraints (18), (19) and (20) can all be removed. Secondly, the capacity constraints (21) are changed to (27), where all replicas are assigned the same amount of CPU power.

$$\sum_{i\mathcal{S}} \sum_{q \in \mathcal{Q}_i} (C_{PUASSiq} VAR_i + C_{PUOHiq}) x_{iqn} - C_{APCPU} o_n \leq 0, \quad \forall n \in \mathcal{N} \tag{27}$$

Then, in order to modify the formulation of the semi-active approach to the capacity-lowering approach, the w_{iqn} variables are still needed, but the v_{iqn} variables are removed, and hence also the constraints (19). Besides, the constraints (20) are altered, giving the constraints (28), which state that a replica of pair (i, q) can only be active on a node if there is a replica of the same pair deployed on that node. In this approach we also use a set of variables m_n, representing the amount of non-assignable CPU power on

node n. These variables should be greater than or equal to the amount of CPU power needed by any passive replica, deployed on the node, in order to turn active. With the introduction of the m_n variables it is also necessary to change the node capacity constraints (21). The new constraints, used to set the m_n variables, and the altered capacity constraints, are depicted in (29) and (30), respectively.

$$w_{iqn} - x_{iqn} \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{28}$$

$$m_n - C_{PUASSiq}VAR_i(x_{iqn} - w_{iqn}) \geq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{29}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUASSiq}VAR_i w_{iqn} +$$
$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUOHiq}x_{iqn} + m_n - C_{APCPU}o_n \leq 0, \quad \forall n \in \mathcal{N} \tag{30}$$

Table 1 sums up the constraints used in the three different approaches. In addition, a summary of the complete models is given in Appendix A.1.

Table 1: Summary of the constraints for the three different approaches to CPU assignment of passive replicas in the SDP-PDC model

Approach	Constraints
Semi-active	(16) - (26)
All-active	(16), (17), (27), (22), (25), (26)
Capacity-lowering	(16) - (18), (28) - (30), (22), (23), (25), (26)

6.4   The Service Deployment Problem in a Hybrid Cloud Environment

This subsection describes a problem which will be referred to as the Service Deployment Problem in a Hybrid Cloud Environment (SDP-HCE), which is an extension of the previous problem. A MILP model of the problem is formulated and the formulation is implemented in commercial software to

be able to test the problem on the provided test instances. Comments on the implementation and presentation of the numerical results are given in Section 7.

6.4.1 Problem definition

Likewise the SDP-PDC, this problem is composed of three main decision problems. The replication level decisions in this problem are made under the same conditions as in the SDP-PDC, whereas the deployment decision environment differ from the previous model. The decisions related to whether a node should be turned off or not are still only considered in the private cloud of the service provider, and hence are not changed. The resulting solutions are evaluated by a cost function which has similarities with the objective function in the SDP-PDC model, but contain additional terms to cater for the change in deployment environment.

Since the replication level decisions and the decisions to turn off nodes are done in the same way as in the previous problem, these decisions are not discussed here. On the other hand, the deployment decisions will now be given an explanation.

Deployment decisions

The active and passive replicas, resulting from a chosen replication pattern, from the set $\mathcal{R}_i$, is to be deployed either on the nodes in the private cloud of the service provider or in the public clouds, thus the deployment environment is considered a hybrid cloud. Replicas mapped to the nodes in the private cloud confront the same constraints as in the previous problem, i.e. replicas of the same service-component pair (i, q) should be deployed node-disjoint, and the deployment of replicas to the nodes faces a capacity constraint. Besides, the nodes are still assumed equal in terms of CPU capacity and power consumption relative to the CPU utilization. The active replicas of a given pair (i, q) should be assigned an amount of CPU power, $C_{PUASSiq}V_{ARi}$, whether it is deployed on a node in the private cloud or in the public cloud. In addition both active and passive replicas still need

an amount of CPU power, C_{PUOHiq} for processing other task that is not directly related to the serving of demand.

Deployment in the public clouds is not capacity-constrained in the same way as in the private clouds. The service provider is assumed of having no control over the physical infrastructure, including the nodes, in the public cloud, but he is able to choose between several geographical locations for deployment in the public clouds. These geographical location, from now on denoted public cloud domains, are defined as the set $\mathcal{D}_{PU}$. In order to deploy replicas in the public clouds, they have to be packaged into VMs, which size and price are decided by the public cloud provider. Hence each public cloud domain is associated with a set of VMs, where the VMs differ in size and hourly cost. Since the replicas needs to be assigned a fixed amount of CPU power, the cheapest VM that can run a given pair (i, q) can be found in advance of the optimization, and thus the cost of deploying a replica in a public cloud domain, $d \in \mathcal{D}_{PU}$, is given as $C_{OSTVMACTiqd}$ and $C_{OSTVMPASiqd}$, respective of whether the replica is considered active or passive. Specifically, this means that all replicas of (i, q) in the all-active approach cost $C_{OSTVMACTiqd}$, while this is also true for both semi-active and active replicas in the semi-active approach.

In addition to node-disjoint deployment on the nodes, some service-component pairs might have requirements on the number of cloud domains to used for deployment of the replicas, as a means to make a service more tolerant to failures in the management system of the cloud domains. The parameters $S_{PREADiq}$ regulate the minimum number of cloud domains used for deployment of the replicas of pair (i, q), and are given as a value between zero and one, where taking the value one means that no replicas should be deployed in the same cloud domain. Lastly, a given component may also have a binding stating that the replicas of the component must be deployed in a specific domain. The parameter which indicate a such binding, B_{INDiq}, for a given pair (i, q), equals d if only domain d might be used for deployment of the replicas of (i, q). If no such binding exists, B_{INDiq} equal zero. Note that a service-component can not have both constraints requiring a minimum number of domains used for deployment and constraints requiring that all replicas should be deployed in a specific domain.

6.4.2 MILP formulation of the SDP-HCE

Sets

As in previous model, we have a set of services, $\mathcal{S}$, a set of components, $\mathcal{Q}_i$, corresponding to a given service i and a set of replication patterns, $\mathcal{R}_i$, for each service i. Moreover the hybrid cloud environment is composed of a set of cloud domains, denoted $\mathcal{D}$, where the public and private cloud domains are characterized by the sets $\mathcal{D}_{PU}$ and $\mathcal{D}_{PR}$, respectively. All private cloud domains are operated by the service provider, whereas the public cloud domains might be operated by different IaaS cloud providers and different IaaS cloud providers might operate several domains. Lastly, $\mathcal{N}_d$ refers to the set of nodes in a given private cloud domain d. A summary of the sets is given below.

$\mathcal{S}$	Set of all services, indexed by i
$\mathcal{Q}_i$	Set of components of service i, indexed by q
$\mathcal{R}_i$	Set of replication patterns for service i, indexed by r
$\mathcal{D}$	Set of cloud domains for deployment, indexed by d
$\mathcal{D}_{PR}$	Set of private cloud domains for deployment, indexed by d
$\mathcal{D}_{PU}$	Set of public cloud domains for deployment, indexed by d
$\mathcal{N}_d$	Set of nodes for deployment in a private cloud domain d, indexed by n

Parameters

The types of parameters used in the private data center model are also included here. In addition, the binding parameters and the parameters stating the cost of deployment in the public cloud domains, as well as the parameters used to control the minimum number of used domains, are introduced. Note that, not all parameters are used in all approaches.

H_{RS}	The length of the time period, in hours
C_{OSTPWR}	Cost per unit of energy usage
$P_{WRCOEFF}$	Coefficient translating CPU utilization to power usage
P_{WRIDLE}	The power consumption of a node in idle state
D_{EMi}	The avg. number of request for service i per time unit
$J_{OBLOADiq}$	The avg. CPU power needed to handle a request in (i, q) per time unit
$C_{PUDEMiq}$	The CPU power demanded by service requests for (i, q) $= D_{EMi} J_{OBLOADiq}$
$C_{PUASSiq}$	The amount of CPU power assigned to an active replica of (i, q)
V_{ARi}	The parameter ensuring that a service can handle the peaks in demand
C_{PUOHiq}	CPU power overhead for management-related tasks in replicas
C_{APCPU}	CPU capacity on each node in a private cloud domain
N_{Riqr}	The total number of replicas of (i, q) using replication pattern r
N_{RMXiq}	The maximum of N_{Riqr} over r
A_{CTRiqr}	The number of active replicas of (i, q) using replication pattern r
$A_{CTRMXiq}$	The maximum of A_{CTRiqr} over r
$S_{ACOEFFiq}$	Coefficient used to control the number of semi-active replicas of (i, q)
$C_{OSTVMACTiqd}$	Cost of using a VM instance for the active replicas of (i, q) in public cloud domain d per hour
$C_{OSTVMPASiqd}$	Cost of using a VM instance for the passive replicas of (i, q) in public cloud domain d per hour
$S_{PREADiq}$	The required number of cloud domains used for a replica in percentage of the total number of replicas
B_{INDiq}	Equal d if pair (i, q) only can be deployed in cloud domain d, and equal 0 if there is no binding on the domains used for deployment.

Variables

Likewise in the previous model, the binary variables y_{ir} is used to indicate whether replication pattern r is employed for service i or not. Since the deployment environment is changed to include several cloud domains, the deployment variables are altered. Firstly, I define u_{iqd} to be binary variables stating whether one or more replicas of pair (i, q) are deployed in cloud domain $d \in \mathcal{D}$. Besides, the integer variables x_{Diqd}, w_{Diqd} and v_{Diqd} denotes the total number of replicas, active replicas and semi-active replicas of pair (i, q) in cloud domain d. In addition, the binary variables x_{iqdn}, w_{iqdn} and v_{iqdn} indicate whether a replica, active replica and semi-active replica of pair (i, q) is deployed on node n in private cloud domain $d \in \mathcal{D}_{PR}$ or not. The variables f_{iqd} are the fraction of demand-serving replicas of pair (i, q) which is deployed in private cloud domain d and are used for determining the amount of the requests served in the private cloud domains. Furthermore, the binary variables o_{dn} state whether a node in private cloud domain d is turned on, and hence used for deployment. Lastly, when applying the capacity-lowering approach, the variables m_{dn} give the amount of non-assignable CPU on node n in private cloud domain d.

$$u_{iqd} = \begin{cases} 1 & \text{if the a replica of } (i, q) \text{ is deployed in cloud domain } d \\ 0 & \text{otherwise} \end{cases}$$

x_{Diqd} The total number of replicas of the pair (i, q) that is deployed in cloud domain d

$$x_{iqdn} = \begin{cases} 1 & \text{if the a replica of } (i, q) \text{ is deployed on node } n \text{ in private cloud domain } d \\ 0 & \text{otherwise} \end{cases}$$

w_{Diqd} The number of active replicas of (i, q) that is deployed in cloud domain d

$$w_{iqdn} = \begin{cases} 1 & \text{if an active replica of } (i, q) \text{ is deployed on node } n \text{ in private cloud domain } d \\ 0 & \text{otherwise} \end{cases}$$

v_{Diqd} The number of semi-active replicas of (i, q) deployed in cloud domain d

$$v_{iqdn} = \begin{cases} 1 & \text{if a semi-active replica of } (i,q) \text{ is deployed on node } n \text{ in} \\ & \text{private cloud domain } d \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ir} = \begin{cases} 1 & \text{if replication pattern } r \text{ is used for service } i \\ 0 & \text{otherwise} \end{cases}$$

$$o_{dn} = \begin{cases} 1 & \text{if node } n \text{ in private cloud domain } d \text{ is turned on} \\ 0 & \text{otherwise} \end{cases}$$

m_{dn} The amount of non-assignable CPU power on node n in private cloud domain $d \in \mathcal{D}_{PR}$, in the capacity-lowering approach

f_{iqd} The fraction of demand-service replicas of (i,q) that is deployed in private cloud domain d

Objective function

The objective function in the SDP-HCE contains two different types of costs, namely the cost of energy usage in the private cloud domains and the cost of deploying the replicas in the public cloud domains. The former costs remain almost the same compared to the cost function in the SDP-PDC, but must be altered to account for the possibility that not all demand are served from the private cloud. Thus the cost of energy usage, z_{EC}, is calculated in the following way.

$$\begin{aligned} z_{EC} = C_{OSTPWR} H_{RS} \Bigg(& \sum_{d \in \mathcal{D}_{PR}} \sum_{n \in \mathcal{N}_d} P_{WRIDLE} o_{dn} \\ & + P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PR}} \sum_{n \in \mathcal{N}_d} C_{PUOHiq} x_{iqdn} \\ & + P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PR}} C_{PUDEMiq} f_{iqd} \Bigg) \end{aligned} \qquad (31)$$

where the last term in (31) accounts for the CPU power needed for serving demand in the private cloud domains, and f_{iqd} is defined by the set of equations (32), in the semi-active and capacity-lowering approach, and (33),

in the all-active approach.

$$f_{iqd} = \sum_{r \in \mathcal{R}_i} \frac{w_{Diqd}}{A_{CTRiqr}} y_{ir}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR} \tag{32}$$

$$f_{iqd} = \sum_{r \in \mathcal{R}_i} \frac{x_{Diqd}}{N_{Riqr}} y_{ir}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR} \tag{33}$$

The deployment costs in the public cloud domains consist of two terms, the cost of deploying active replicas and the cost of deploying passive replicas. This part of the objective function is different in the various approaches for assignment of CPU power to the passive replicas. In the all-active approach all replicas of (i, q) costs $C_{OSTVMACTiqd}$ for deployment in public cloud domain d, while the same is true for active and semi-active replicas in the semi-active approach. Lastly, the active and passive replicas in the capacity-lowering approach costs $C_{OSTVMACTiqd}$ and $C_{OSTVMPASiqd}$ for deployment in public cloud domain d, respectively. The total, hourly deployment costs of all approaches, z_{DCAA}, z_{DCSA} and z_{DCCL}, are given by (34), (35) and (36), for the all-active, semi-active and capacity-lowering approach, accordingly.

$$z_{DCAA} = \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PU}} C_{OSTVMACTiqd} x_{Diqd} \tag{34}$$

$$z_{DCSA} = \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PU}} C_{OSTVMACTiqd}(w_{Diqd} + v_{Diqd})$$
$$+ \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PU}} C_{OSTVMPASiqd}(x_{Diqd} - (w_{Diqd} + v_{Diqd})) \tag{35}$$

$$z_{DCCL} = \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PU}} C_{OSTVMACTiqd} w_{Diqd}$$
$$+ \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PU}} C_{OSTVMPASiqd}(x_{Diqd} - w_{Diqd}) \tag{36}$$

This leads to three different objective functions, z_{HCEAA}, z_{HCESA} and z_{HCECL}, shown in (37), (38) and (39), where the hourly cost of using the public cloud domains is multiplied with the length of the time period.

$$\min z_{HCEAA} = z_{EC} + H_{RS} z_{DCAA} \tag{37}$$

$$\min z_{HCESA} = z_{EC} + H_{RS} z_{DCSA} \tag{38}$$

$$\min z_{HCECL} = z_{EC} + H_{RS} z_{DCCL} \tag{39}$$

Constraints

Likewise when the constraints in the SDP-PDC model was presented, I begin by explaining the constraints in the semi-active approach and the continue with describing the necessary adjustments done in order to model the all-active and capacity-lowering approach.

The set of constraints ensuring that only one replication pattern is chosen for each service is the same as in the SDP-PDC model.

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1, \quad \forall i \in \mathcal{S} \tag{40}$$

To guarantee that all replicas of a pair (i, q) is deployed either on the nodes in the private cloud or in the public cloud, I firstly choose to distribute the replicas to the private and public domains, and further map the replicas deployed in a private domain to the nodes. The same is done when the decisions about setting replicas to be active or semi-active. This is shown in (41) - (46). In the same manner as in the SDP-PDC model, one must include a set of constraints ensuring that pair (i, q) only can be active or semi-active replica on a node if a replica is deployed there. The same have to be done on domain level in the public cloud domains, i.e. the number of active and semi-active replicas of pair (i, q) in public cloud domain d

cannot be greater than the total number of replicas in this domain. This is done in constraints (47) and (48).

$$\sum_{d \in \mathcal{D}} x_{Diqd} - \sum_{r \in \mathcal{R}_i} N_{Riqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{41}$$

$$\sum_{n \in \mathcal{N}_d} x_{iqdn} - x_{Diqd} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR} \tag{42}$$

$$\sum_{d \in \mathcal{D}} w_{Diqd} - \sum_{r \in \mathcal{R}_i} A_{CTRiqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{43}$$

$$\sum_{n \in \mathcal{N}_d} w_{iqdn} - w_{Diqd} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR} \tag{44}$$

$$\sum_{d \in \mathcal{D}} v_{Diqd} - $$
$$\sum_{r \in \mathcal{R}_i} \left\lceil S_{ACOEFFiq} \frac{N_{Riqr} - A_{CTRiqr}}{A_{CTRiqr}} \right\rceil y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{45}$$

$$\sum_{n \in \mathcal{N}_d} v_{iqdn} - v_{Diqd} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR} \tag{46}$$

$$w_{Diqd} + v_{Diqd} - x_{Diqd} \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PU} \tag{47}$$

$$w_{iqdn} + v_{iqdn} - x_{iqdn} \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{48}$$

As mentioned, some service components may have requirements on the number of cloud domains used for deployment by the replicas. The parameters $S_{PREADiq}$ are used to control the number of domains used, and take values between zero and one. Constraints (49) model these requirements. The variables u_{iqd} are used to keep track of the domains where a pair (i, q) has replicas deployed, and thus one has to prevent these variables of being one if the domain is not used. (50) prevent such cases.

$$\sum_{d \in \mathcal{D}} u_{iqd} - \sum_{r \in \mathcal{R}_i} N_{Riqr} S_{PREADiq} y_{ir} \geq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{49}$$

$$x_{Diqd} - u_{iqd} \geq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D} \tag{50}$$

The capacity constraints on the nodes in this model is equal to the capacity constraints in the previous, except that all variables are indexed with a private cloud domain index, d, as well.

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUASSiq} V_{ARi} w_{iqdn} +$$
$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUASSiq} V_{ARi} v_{iqdn} +$$
$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUOHiq} x_{iqdn} - C_{APCPU} o_{dn} \leq 0, \quad \forall d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{51}$$

According to the first part of the objective function, it is necessary to determine the fraction of the demand for pair (i, q) that is served in a private cloud domain d, i.e. determine the f_{iqd} variables. (32) shows a non-linear equation for setting these variables, and hence to be able to keep the model linear that equation must be altered. Therefore (32) is transformed to the set of inequalities in (52). Note that the inequalities, for a given combination of i, q and d, will be redundant for all r except for the one where $y_{ir} = 1$.

$$\frac{w_{Diqd}}{A_{CTRiqr}} - f_{iqd} -$$
$$\frac{A_{CTRMXiq}}{A_{CTRiqr}} (1 - y_{ir}) \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, r \in \mathcal{R}_i \tag{52}$$

There exist also service components that, because of for example security issues, have bindings on which cloud domain that can be used for deployment (i.e. $B_{INDiq} = d$). In order to handle this, I introduce constraints forcing a x_{Diqd} variable to zero if deployment of a replica of (i, q) is prohibited in domain d. On the other hand, I set a x_{Diqd} variable to be greater than or equal to one if deployment of a replica of (i, q) is only allowed in domain d. If there is no bindings on the domains used for deployment of a replica (i.e. $B_{INDiq} = 0$), I treat the x_{Diqd} variable as positive

integers. However, the constraints (41) will make x_{Diqd} take an integer value if $B_{INDiq} \neq 0$, as well. This gives:

$$x_{Diqd} \in \mathbb{N}, \quad \forall i \in \mathcal{S}, q \in \{q' \in \mathcal{Q}_i : B_{INDiq'} = 0\}, d \in \mathcal{D} \qquad (53)$$

$$x_{Diqd} \geq 1, \quad \forall i \in \mathcal{S}, q \in \{q' \in \mathcal{Q}_i : B_{INDiq'} \neq 0\},$$
$$d \in \{d' \in \mathcal{D} : B_{INDiq} = d'\} \qquad (54)$$

$$x_{Diqd} = 0, \quad \forall i \in \mathcal{S}, q \in \{q' \in \mathcal{Q}_i : B_{INDiq'} \neq 0\},$$
$$d \in \{d' \in \mathcal{D} : B_{INDiq} \neq d'\} \qquad (55)$$

Lastly, we restrict the rest of the variables, expect f_{iqd}, to be binary or integer.

$$w_{Diqd} \in \mathbb{N}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D} \qquad (56)$$

$$v_{Diqd} \in \mathbb{N}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D} \qquad (57)$$

$$u_{iqd} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D} \qquad (58)$$

$$x_{iqdn} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \qquad (59)$$

$$w_{iqdn} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \qquad (60)$$

$$v_{iqdn} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \qquad (61)$$

$$y_{ir} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, r \in \mathcal{R}_i \qquad (62)$$

$$o_{dn} \in \{0, 1\}, \quad \forall d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \qquad (63)$$

In order to adjust the model above to the all-active approach one must remove and change some constraints. Firstly, as all replicas are assumed active it is not necessary to use the w_{Diqd}, w_{iqdn}, v_{Diqd} and v_{iqdn} variables, and thus the constraints (43) through (48) are removed. Thereafter, the capacity constraints (51) are altered to (64), in the same manner as in the SDP-PDC model. Besides, the constraints setting the f_{iqd} variables are altered to (65), since all replicas are active. The rest of the model remains the same.

$$(C_{PUASSiq}V_{ARi} + C_{PUOHiq})x_{iqdn} -$$
$$C_{APCPU}o_{dn} \leq 0, \quad \forall d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \qquad (64)$$

$$\frac{x_{Diqd}}{N_{RMXiq}} - f_{iqd} -$$

$$\frac{N_{RMXiq}}{N_{Riqr}}(1 - y_{ir}) \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, r \in \mathcal{R}_i \qquad (65)$$

Next, to alter the model of semi-active approach to the capacity-lowering approach some modifications has to be done. Likewise, in the modification process above, some variables can be removed, namely the v_{Diqd} and v_{iqdn} variables. This leads to the removal of constraints (45) and (46), and adjustment of (47) and (48) to (66) and (67) below. Furthermore, we have to set the variables, m_{dn}, defining the amount of non-assignable CPU power on each node in the private cloud domains, and thus also alter the capacity constraints on the nodes. (68) and (69) set the m_{dn} variables and constrains the CPU power assignment on the nodes, in the same manner as in the SDP-PDC model.

$$w_{Diqd} - x_{Diqd} \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PU} \qquad (66)$$

$$w_{iqdn} - x_{iqdn} \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \qquad (67)$$

$$m_{dn} -$$
$$C_{PUASSiq}V_{ARi}(x_{iqdn} - w_{iqdn}) \geq 0, \ \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \quad (68)$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUASSiq}V_{ARi}w_{iqdn} +$$
$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUOHiq}x_{iqdn} - C_{APCPU}o_{dn} \leq 0, \quad \forall d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \qquad (69)$$

Table 2 concludes the presentation of the mathematical models by giving a summary of the constraints used in the three different approaches of CPU assignment to passive replicas. A summary of the models is also given in Appendix A.2

Table 2: Summary of the objective function and the constraints for the three different approaches to CPU assignment of passive replicas in the SDP-HCE model

Approach	Objective	Constraints
Semi-active	(38)	(40) - (63)
All-active	(37)	(40) - (42), (49), (50), (64), (65), (53) - (55), (58), (59), (62), (63)
Capacity-lowering	(39)	(40) - (44), (66), (67), (49), (50), (68), (69), (52) - (56), (58) - (60), (62), (63)

6.5 Generation of replication patterns

So far we have ignored the generation of the set of replication patterns for each service. The simplest way of generating these is by a total enumeration of all combinations of active and passive replicas for each component of a service, while simultaneously giving an upper bound on the total number of replicas. Thereafter one could check if a pattern confirms with the response time and availability requirements. If the response time requirement of a service is set high and the availability requirement is set low, this would lead to a great number of patterns. But by analysing the problem it is possible to remove several patterns that satisfy the SLA requirements as well. Besides, by noticing that only the number of active replicas affect the response time (cf. (9)) and the total number of replicas compared to the number of active replicas affect the availability (cf. (10)), it is possible to construct an algorithm consisting of an outer and an inner loop. The outer loop constructs possible combinations of the number active replicas of the components of a given service, while the inner loop uses the resulting number of active replicas, produced in the outer loop, and adds a combination of the number of passive replicas of each component to the pattern, in order to make an SLA-corresponding pattern. For the sake of readability of the rest of this section, I define AC as a combination of the number of active replicas for each component of a given service. Likewise, I define TC as a combination of the total number of replicas for each component of a given service, and together, AC and TC forms a replication pattern.

Algorithm 6.1, on page 62, shows the main parts of the approach taken

to generate replication patterns in my models. The algorithm takes a given service i as input and uses the *global* parameters for the upper bound on the response time, the lower bound on the availability and the maximum number of active replicas and the maximum total number of replicas for each component of i. The two latter parameters is set high enough in order to not constrain the MILP solution. When finished running, the algorithm have created the set of replication patterns and the matrices containing the number of active replicas, A_{CTRiqr}, and the total number of replicas, N_{Riqr} for each component of i.

Firstly, the algorithm initialize the temporary number of active replicas, T_{MPAq}, of each component of i and their upper limits. The initialization sets the number of active replicas to one, but this number could of course be increased if one has some preferences or knowledge of the problem in advance. After the initialization, the algorithm enters the outer loop where the AC is controlled. If the combination passes the control, then the temporary number of total replicas, T_{MPNq}, and the upper limits, are initialized for each component of i, and the algorithm proceeds to the inner loop. In the inner loop, the pattern, consisting of an AC and a TC, is controlled. If the pattern passes the control, then the A_{CTRiqr} and N_{Riqr} matrices and the set of replication patterns, $\mathcal{R}_i$, are updated with the new pattern and r is incremented. Thereafter, the algorithm will generate more TCs, and update the matrices and the set of replication patterns when a pattern passes the control. This loop will continue until T_{MPNq} reaches its upper limit for all components, q, of i. When the inner loop has finished, there are generated a new AC, and this new combination is sent to the inner loop if it passes the control. Likewise the inner loop, the outer loop will continue until T_{MPAq} reaches its upper limit for all components, q, of i.

So far I have not explained what is controlled in the outer and inner loop. Both controls consist of two tests, and the first test in the outer loop checks the response time requirement, while the first test in the inner loop checks the availability requirement of the service. The second tests in both loops are more complex, but will be explained now, through an example. Let us assume that a service consists of three components, and after running the algorithm for a couple of iterations in the outer loop, we know

that a solution with two, three and two active replicas of the components agrees with the response time requirement. I denote this AC by (2,3,2), for simplicity. Furthermore, there exists at least one TC, which together with (2,3,2) forms a pattern that satisfies the availability requirement. Now, assuming that there is generated an AC with the numbers (2,3,3), we know that this combination satisfies the response time requirement, but it is a more expensive solution, and hence we discard it and do not proceed to the inner loop. (2,3,3) is a more expensive combination than (2,3,2) as the last component has a higher number of active replicas while the first and second have an equal number of active replicas, and thus the cost minimization would prefer (2,3,2) above (2,3,3). However, if a combination of active replicas with the numbers (1,3,3) is produced and the combination satisfies the response time requirement, we can not directly tell whether it is a more expensive solution than (2,3,2) or not, and therefore I choose to proceed with such events. A generalization of this is to send a combination of the number of active replicas to the inner loop if it satisfies the response time requirement and has a lower number of active replicas, for at least one component, than a already found pattern in $\mathcal{R}_i$. The second test in the inner loop is based on the same line of thought, but one must only compare a TC with a already found pattern in $\mathcal{R}_i$ that is based on the same AC, cf. $\mathcal{R}_i^*$ in Algorithm 6.1.

Algorithm 6.1: CREATEREPLICATIONPATTERNS(i)

global R_{EQRTi}	The required response time for service i
global R_{EQAVi}	The required availability for service i
global $M_{AXREPiq}$	Maximum number of replicas for (i, q)
global $M_{AXACTiq}$	Maximum number of active replicas for (i, q)
global $\mathcal{R}_i$	The set of replication patterns for i, to be calculated
global N_{Riqr}	The number of replicas given r, to be calculated
global A_{CTRiqr}	The number of active replicas given r, to be calculated
local T_{MPNq}	The temporary total number of replicas of component q
local T_{MPAq}	The temporary number of active replicas of component q
local N_{UPq}	The temporary upper bound of number of replicas in total for q
local A_{UPq}	The temporary upper bound of number of active replicas for q
local r	The replication settings counter

comment: Initialize r for service i

$r \leftarrow 1$

comment: Initialize the number of active replicas

for each $q \in \mathcal{Q}_i$

$\quad$ **do** $\begin{cases} T_{MPAq} \leftarrow 1 \\ A_{UPq} \leftarrow M_{AXACTiq} \end{cases}$

while $\exists\, q \in \mathcal{Q}_i : T_{MPAq} < A_{UPq}$

$\quad$ **do** $\begin{cases} \textbf{if } rt(T_{MPA}, i) \leq R_{EQRTi} \textbf{ and } \exists\, r \in \mathcal{R}_i, q \in \mathcal{Q}_i : T_{MPAq} < A_{CTRiqr} \\ \\ \textbf{then} \begin{cases} \textbf{comment: Initialize the total number of replicas} \\ \\ \textbf{for each } q \in \mathcal{Q}_i \\ \quad \textbf{do } \begin{cases} T_{MPNq} \leftarrow T_{MPAq} \\ N_{UPq} \leftarrow M_{AXREPiq} \end{cases} \\ \textbf{while } \exists\, q \in \mathcal{Q}_i : T_{MPNq} < N_{UPq} \\ \quad \textbf{do} \begin{cases} \textbf{if } av(T_{MPA}, T_{MPN}, i) \geq R_{EQAVi} \textbf{ and} \\ \quad \exists\, r \in \mathcal{R}_i^{*\,a}, q \in \mathcal{Q}_i : T_{MPNq} < N_{Riqr} \\ \\ \textbf{then} \begin{cases} \textbf{comment: Update } N_R \textbf{ and } A_{CTR} \\ \\ \textbf{for each } q \in \mathcal{Q}_i \\ \quad \textbf{do } \begin{cases} N_{Riqr} \leftarrow T_{MPNq} \\ A_{CTRiqr} \leftarrow T_{MPAq} \end{cases} \\ \textbf{comment: Update } \mathcal{R}_i \textbf{ and increment } r \\ \text{Add } \{r\} \text{ to } \mathcal{R}_i \\ r \leftarrow r + 1 \end{cases} \\ \textbf{comment: Find new } T_{MPN} \\ T_{MPN} \leftarrow findNewTMPN(T_{MPN}) \end{cases} \end{cases} \\ \textbf{comment: Find new } T_{MPA} \\ T_{MPA} \leftarrow findNewTMPA(T_{MPA}) \end{cases}$

$^{a}\mathcal{R}_i^* = \{ r \in \mathcal{R}_i \mid \forall\, q \in \mathcal{Q}_i : T_{MPAq} = A_{CTRiqr} \}$

7 Numerical results and discussion

The primary objective of the models developed in this thesis are to act as decision support for service providers when making their decisions about service deployment and replication levels. In order to investigate the applicability of the models, they are tested on different test instances. Firstly, I will give some comments on the implementation and the generation of the test instances. Thereafter, the results obtained on the smallest test instances is detailed, before the modelled on larger test instances in order to discuss the scalability of the models. Lastly, some performance and dependability concerns is discussed.

7.1 Implementation

To be able to test the different models using the test instances, to be presented, both models are implemented in the Xpress Mosel modelling language (version 3.2.0) and run in the Xpress-IVE (version 1.21.02) environment with the Xpress Optmizer (version 21.01.00) solver. The computer running this software has a Intel Core 2 Duo E6700 (2x2.6GHz) processor and 4GB of RAM. All Mosel code can be found in Appendix B.

Since the model is implemented in a modelling language, there might exist language specific features that can be utilized in order to make the model more efficient, in terms of complexity and solution time. In the implementation I have taken advantage of the option to dynamically create variables, which slightly simplifies the formulation. This will be described in the next paragraph. Besides, I have included some extra constraints in the implementation with the purpose of removing symmetrical solutions. These constraints will be elaborated thereafter.

Modelling of the constraints binding replicas to a specific domain

In the model formulation of the SDP-HCE model, in Section 6.4.2, the constraints forcing all replicas of a given component to be deployed in a given domain are handled by setting the x_{Diqd} variables to zero for prohibited domains. However, in the implementation of the model in the Mosel

modelling language, the formulation of these constraints is done by dynamically creating the deployment variables for domains that might be used by a component for deployment. Hence one avoid creating variables that, when set to another value than zero, nevertheless will be infeasible.

Additional constraints in the implementation

As a reason of the fact that all nodes are modelled equal in terms of CPU capacity and power consumption, relative to the utilization, solving the implementation of the model generates symmetrical solutions. Specifically, this means that the optimal value does not change if all replicas deployed on one node switch place with all replicas deployed on another node. Likewise, if one node out of the total nodes is turned off, the objective value is independent of which node that is turned off. Hence, several nodes in the branch and bound (B&B) tree, not to be confused with the physical machines in the models, will contain equally optimal solutions, which may lead to slower convergence of the branch and bound algorithm.

In order to reduce the number of symmetrical solutions I choose to sort the nodes according to the amount of their capacity which is assigned to replicas. This means that the node with the lowest index number has the greatest amount of its capacity assigned to replicas, while the node with the highest index number has the least amount of its capacity assigned to replicas, and hence also make potential turned off nodes being the ones with the highest index number. To be able to do this sorting, I define a variable, s_n, for each node n, which equal the amount of capacity on a node that is assigned to replicas. This is done mathematically as in (70) for the capacity-lowering approach, and in the correspondingly for the other two approaches. (71) handles the sorting.

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUASSiq} V_{ARi} w_{iqn} +$$
$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUOHiq} x_{iqn} - s_n = 0, \quad \forall n \in \mathcal{N} \tag{70}$$

$$s_n - s_{n+1} \geq 0, \quad \forall n \in \{n' \in \mathcal{N} : n' < |\mathcal{N}|\} \tag{71}$$

Constraints (70) and (71) remove otherwise, feasible solutions without affecting the optimal value. But including these constraints in the model, also have a drawback. In preliminary tests, it turns out that the feasible set has been so reduced that the Xpress solver has difficulties to find even a feasible solution in the larger test instances. In order to overcome this, I have implemented another less strict set of constraints, shown in (72). These constraints simply sort the nodes according to whether they are turned off or not.

$$o_n - o_{n+1} \geq 0, \quad \forall n \in \{n' \in \mathcal{N} : n' < |\mathcal{N}|\} \tag{72}$$

In conjunction with (72), I choose to deploy some active replicas of a selected service-component pair before the optimization is called. The selected service-component pair is the one with highest number of active replicas in the replication pattern with the lowest number of active replicas. Mathematically, let us denote this number of active replicas by A, then A equal $\max_{(i,q)} \min_r \{A_{CTRiqr}\}$. These A active replicas is consequently deployed on the A nodes with the lowest index numbers. This predeployment strategy will not affect the optimal solution when (72) is used. However, it can not be combined with (70) and (71), since we can not know if these A active replicas will be deployed on the nodes with the greatest amount of CPU power assigned to the replicas, in the optimal solution. This predeployment strategy was also used in (Gullhav, 2010), where there was shown that the strategy gave shorter solution times. Using (72), together with the predeployment strategy, instead of (70) and (71), might still give some symmetrical solutions, but makes it easier for the Xpress solver to discover solutions.

The additional constraints (70) - (72) is based on the SDP-PDC model, but is also applied on the SDP-HCE model, where the s_n and o_n variables are formulated with an index, d, for the private cloud domain corresponding to the node n. In order to do use the predeployment strategy in a private cloud domains in the SDP-HCE model, I only consider predeployment of

Table 3: Overview of the strategies to reduce the number of symmetrical solutions

Name	Constraints/strategy
SYM0	Predeployment strategy
SYM1	Predeployment strategy and (72)
SYM2	(70) and (71)

replicas of a service component which are forced to be deployed in a private domain, i.e. replicas with $B_{INDiq} = d$, $d \in \mathcal{D}_{PR}$.

In the testing of the models I have, in addition to the two preceding strategies, also included a strategy which only considers predeployment of replicas. The three strategies are denoted SYM0, SYM1 and SYM2, and Table 3 gives an overview of the three.

7.2   Generation of test instances

Some of the data used in the generation of test instances are found by doing web searches, while other data is randomly generated within an interval with approximated bounds. All data used in the models are presented below.

Problem scales

The test instances for the SDP-PDC model range from a case with five services and twelve nodes to a case with forty services and eighty nodes. In the SDP-HCE test instances I have only tested the model for one private cloud domain, with varying number of nodes, and two public cloud domains, provided by two different cloud IaaS providers. The number of components of the services is generated by the use of a function which produces random numbers. The minimum number of components is set to two, while the maximum is set to four, in all test instances, expect for one test instance where the maximum total number is raised to five. In the generation of replication patterns each service component is associated with a maximum

Table 4: Scale of the different test instances

Test instance name	# services	range of # component	# nodes	# public domains
PDC-5	5	2-4	12	
PDC-15	15	2-5	35	
PDC-20	20	2-4	40	
PDC-40	40	2-4	80	
HCE-5	5	2-4	5	2
HCE-10	10	2-4	12	2
HCE-20	20	2-4	25	2

number of replicas and maximum number of active replicas. These numbers are accordingly set to ten and seven.

Table 4 shows the scale of the different test instances. The naming of the test instances, used in the table, will be retained throughout the thesis.

Availability and response time

Each service is associated with a lower bound on availability and an upper bound on response time, and these values are generated randomly. The response time bound of a service, R_{EQRTi}, is a random number between 200ms and 400ms multiplied by the number of components of this service. The availability bound, R_{EQAVi}, of a service is a random number between 0.98500 and 0.99999. Besides, the asymptotic availability of a service component, $A_{VAILCiq}$, is generated randomly and takes a value between 0.950 and 0.999. Lastly, some components in the SDP-HCE model has restrictions on the minimum number of domains used, compared to the number of replicas deployed, given by $S_{PREADiq}$. This parameter takes a random value between 0 and 0.4.

Demand and resource consumption

The average number of arriving requests per second, D_{EMi}, for each service, i, and thus also for each component of i, is a randomly generated integer between 20 and 40. Moreover, the parameters V_{ARi} of a service, i, is also randomly generated and takes a value between 1 and 1.5. The amount of

CPU power per request per second needed to process a request in service-component pair (i, q), $J_{OBLOADiq}$ is random number between 0.25 and 1, in percentage of the nodes' CPU capacity. Lastly, the average service rate of requests, usually denoted μ in queueing theory, of a service-component pair (i, q) is random number between 0.2 and 1.5 multiplied by the average number of arriving requests per second. In this manner, without consideration V_{ARi}, the fixed amount of CPU power allocated to a replica of (i, q), denoted $C_{PUASSiq}$, equals $\mu_{iq} J_{OBLOADiq}$.

The B_{INDiq} parameters

Some service components in the SDP-HCE model is bound to have its replicas in a specific domain. I have only tested cases where the B_{INDiq} parameter takes a value of one or zero, meaning that no test instance has a component forced to have all its replicas deployed in a public cloud domain.

Power consumption

In accordance with the relationship between utilization and power consumption, (1), I have to set values for the power consumption of a node in idle state and the power consumption of a fully utilized node. Fan et al. (2007) report a nameplate power of 250W for a typical server, and I choose to use this value as the power consumption of a fully utilized node. Beloglazov et al. (2011) and Barroso and Hölzle (2007) state that an idle server consumes around 70 percent and 50 percent of the peak power, respectively, and hence I set the idle power of a node to 150W. Furthermore, I have scaled these numbers to kilowatt as electricity prices are given in ¢ per kilowatt-hour. Thenceforth, I use the peak power consumption and the idle power consumption to calculate the $P_{WRCOEFF}$ parameter, which give $\frac{0.25-0.15}{100\%} = 0.001$[2]. Since the CPU capacity of the nodes are normalized to 100 percent it is necessary to divide the coefficient by 100.

---

[2]Unfortunately, a huge mistake has been done when setting $P_{WRCOEFF}$ in the test instances. An human error, by me, led this parameter to be set to 0.0001. This was discovered too late before the deadline, and thus the numbers presented are erroneous.

Costs

Firstly, the cost of energy usage, C_{OSTPWR}, is set to 10 ¢ per kilowatt-hour, which is an approximate price for electricity costs in the industrial sector in the USA (EIA, 2011). Furthermore, the cost of deploying replicas in the public cloud domains is based on prices from Amazon AWS (Amazon EC2 website, 2011) and Rackspace (Rackspace Cloud hosting website, 2011). These IaaS cloud providers provide several types of VMs with varying size and price. I focus on the on-demand-based, standard instances, and in this category Amazon AWS provide three different types of VMs, while Rackspace provide seven. Amazon presents the CPU sizes of their VM types using elastic compute units (ECU), and in order to include these VM types in the modelling, I have done an estimation on the relative size of these elastic compute units compared to the nodes in the private cloud. Table 5 shows the hourly cost, CPU size, relative to the nodes, and memory size for the VM types of Amazon. Rackspace, on the other hand, does not provide information of about the CPU sizes of its VM types. So in order to set the CPU sizes of Rackspace's VM types, I have compared the VM types of Amazon and Rackspace according to the size of memory and based the estimation of the CPU sizes on this comparison. Table 6 shows the price, memory size and estimated CPU size of Rackspace's VM types. In my test instances the public cloud domain based on Amazon's VM instances is denoted with domain index 2, while the public cloud based on Rackspace's VM instances is denoted with domain index 3. The private cloud domain is denoted with index 1. Lastly, for the sake of simplicity, I set the H_{RS} parameter to 1, which has no impact on the solution in none of the models.

From the Tables 5 and 6 and the cost of energy usage, given above, it is clear that deployment in the public cloud domains is much more expensive than deployment in the service provider's own private cloud. In order to be able to weight the benefits of deployment in the public cloud and the private cloud, it is necessary to take into account other costs than only the cost of energy usage in the private cloud, like management and other cost related the physical infrastructure. Such costs might hardly be influenced in a short time period, and is hence not included in this thesis.

Table 5: Amazon standard VM types[1] with corresponding hourly cost, CPU size and memory size (Amazon EC2 website, 2011)

VM type	Cost (¢ per hour)	CPU size (in %)	Memory size (GB)
VM1	9.5	10	1.7
VM2	38	40	7.5
VM3	76	80	15

Table 6: Rackspace standard VM types[2] with corresponding hourly cost, estimated CPU size and memory size (Rackspace Cloud hosting website, 2011)

VM type	Price (¢ per hour)	Estimated CPU size (in %)	Memory size (MB)
VM1	1.5	2	256
VM2	3	5	512
VM3	6	7.5	1024
VM4	12	10	2048
VM5	24	25	4096
VM6	48	40	8192
VM7	96	80	15872

[1]EC2 instance types
[2]Cloud Servers

Infeasible test instances

Since much of the data is randomly generated, generation of infeasible test instances might happen. Test instances might be infeasible if, for example, the $S_{PREADiq}$ parameter is set too high for a service-component with a large number of replicas, i.e. a service component have a lower bound on the number of used domains that is higher than the number of domains. In such cases the test instance is regenerated until a feasible test instances is created.

Validity of test instances and results

A consequence of the fact the much of the data presented above are based on estimation and assumptions, the main focus will not be directly on testing if the proposed models give better results than other models in literature. Instead I will concentrate on testing the applicability of the models, both in

Table 7: PDC-5 *capacity-lowering* approach: response time and availability of each service

Service	Availability	Req. availability	Response time (ms)	Req. response time (ms)
1	0.99378	0.98747	345	560
2	0.99338	0.99279	613	630
3	0.98702	0.98546	791	1020
4	0.99938	0.99810	553	1120
5	0.98808	0.98692	397	1200

terms of whether the models behaves as intended or not and if the models can tackle larger problem sizes.

7.3 Detailed results of the SDP models

Now, I will present the results obtained with the SDP-PDC and SDP-HCE models on the smallest test instances, PDC-5 and HCE-5, correspondingly. Both models are tested using the different approaches for assignment of CPU power to passive replicas.

7.3.1 The PDC-5 test instance

The PDC-5 test instance consists of five service where the number of components of each service is two, three, three, four and four, respectively. These sixteen components are to be replicated into a number of replicas, and each replica is to be deployed on one of the twelve nodes, with the objective of minimizing the total energy costs.

Firstly, I will go through the solutions produced by employing the SDP-PDC model with the capacity-lowering approach on the PDC-5 test instance. In this case the SYM1 strategy is used to reduce the number of symmetrical solutions. The chosen replication level for each component of each service is depicted in Figure 9, where the services are denoted by s1 to s5. The resulting availability and response time of each service is shown in Table 7 together with the corresponding availability and response time requirements.

Table 8 gives the resulting deployment configuration by showing the
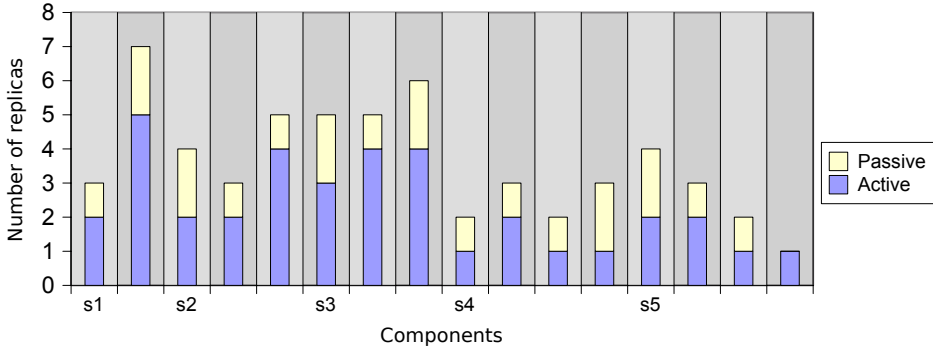
Figure 9: PDC-5 *capacity-lowering* approach: replication levels of each component of each service, denoted by s1 to s5.

CPU assignment of the replicas on the nodes, where a service-component pair is denoted by a (i, q) in the first column. The passive replicas can be distinguished from the active ones, as the passive replicas are only assigned a low amount of CPU power, enough to do tasks related to management and state updating (cf. C_{PUOHiq}). Although, there a twelve nodes which can be used for deployment, an energy-efficient solution will try to shut down some nodes if possible, and thus in this case, only eight nodes are used. Table 8 also shows the amount of non-assignable CPU power at each node, in the second row from the bottom. This number should be greater or equal to the amount of CPU power needed to make any passive replica active in case of a failure. For example, on node 6 this amount equal 31.2, which is the amount of CPU power needed in order to turn the passive replica of (4,4) active. It is worth noticing that all passive replicas are collocated on the same, small subset of nodes, and thus only the four last nodes have a non-assignable amount of CPU power. Figure 10 illustrates the distribution of the amount of CPU power assigned to active and passive replicas, and the non-assignable amount of CPU power on each node. The figure also shows the result of using SYM1 to reduce the number of symmetrical solutions, as the amount of CPU power assigned to the replicas is decreasing with node

Table 8: PDC-5 *capacity-lowering* approach: mapping between replica and node. An assignment is denoted by the amount of CPU power a replica is assigned on a node.

(i,q)	**Nodes** (CPU assignment in %)							
	1	2	3	4	5	6	7	8
(1,1)	22.8			22.8				0.7
(1,2)	11.1	11.1	11.1	0.7	11.1	0.7	11.1	
(2,1)			20.0	20.0	0.8	0.8		
(2,2)	6.8					0.8		6.8
(2,3)	9.0	9.0			0.6	9.0		9.0
(3,1)	17.4			17.4	0.6	0.6		17.4
(3,2)	4.4	4.4			0.8		4.4	4.4
(3,3)	7.9	7.9		7.9		0.7	0.7	7.9
(4,1)				17.6	0.8			
(4,2)			21.0			21.0		0.6
(4,3)			25.0				1.0	
(4,4)					31.7	0.5		0.5
(5,1)	19.0		19.0		0.8	0.8		
(5,2)					33.5	33.5		1.0
(5,3)							51.2	0.5
(5,4)		64.3						
Non-ass	0.0	0.0	0.0	0.0	19.2	31.2	24.0	50.7
Free cap	1.6	3.3	3.9	13.6	0.1	0.4	7.6	0.5

number.

Table 9 gives the actual utilization of the replicas. The utilization of a replica is given by (11), without the summation over services and components, used in the derivation of the objective function in the SDP-PDC model. Hence the utilization is based on both the CPU power needed for serving demand and the CPU overhead. The total utilization on a node is shown in the bottom line, and we can see that the nodes with the most non-assignable CPU power is the ones with the lowest utilization.

Now, the solutions produced by the semi-active approach are considered, and the results are obtained with the SYM2 strategy. In semi-active approach, the choice of a replication pattern also indicates the number of semi-active replicas to deploy. The chosen replication levels in the semi-active approach is depicted in Figure 11, and one can see that the replication levels are not quite the same as in the capacity-lowering approach. While service 2 to service 5 remain unchanged, the first component of service 1 has now one active replica less and the second component has one active
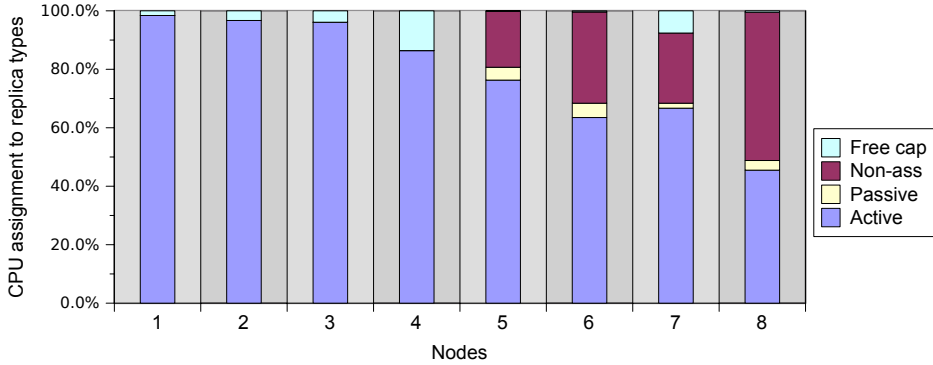
Figure 10: PDC-5 *capacity-lowering* approach: distribution of active and passive replicas and non-assignable CPU power on each node.

Table 9: PDC-5 *capacity-lowering* approach: utilization of the nodes.

(i, q)	Nodes (Utilization of the nodes in %)							
	1	2	3	4	5	6	7	8
(1,1)	8.4			8.4				0.7
(1,2)	5.8	5.8	5.8	0.7	5.8	0.7	5.8	
(2,1)			11.0	11.0	0.8	0.8		
(2,2)	4.6					0.8		4.6
(2,3)	5.2	5.2			0.6	5.2		5.2
(3,1)	9.7			9.7	0.6	0.6		9.7
(3,2)	2.9	2.9			0.8		2.9	2.9
(3,3)	4.6	4.6		4.6		0.7	0.7	4.6
(4,1)				11.1	0.8			
(4,2)			9.2			9.2		0.6
(4,3)			14.6				1.0	
(4,4)					22.3	0.5		0.5
(5,1)	11.1		11.1		0.8	0.8		
(5,2)					19.1	19.1		1.0
(5,3)							26.6	0.5
(5,4)		35.7						
Total	52.4	54.1	51.8	45.6	51.7	38.4	37.1	30.2
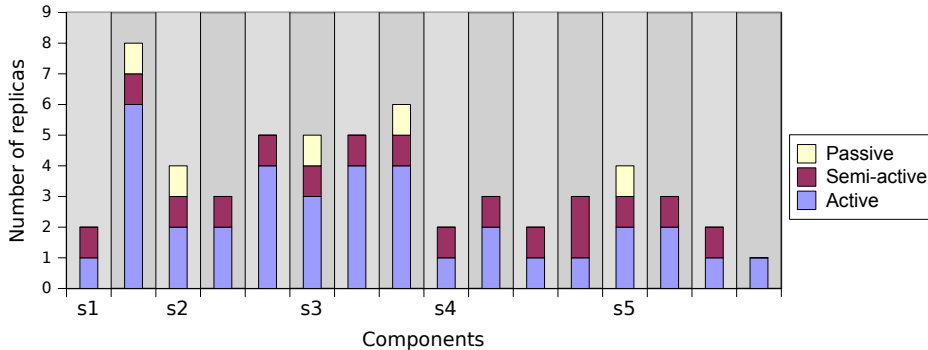
Figure 11: PDC-5 *semi-active* approach: replication levels of each component of each service, denoted by s1 to s5.

Table 10: PDC-5 *semi-active* approach: response time and availability of each service

Service	Availability	Req. availability	Response time (ms)	Req. response time (ms)
1	0.99338	0.98747	557	560
2	0.99338	0.99279	613	630
3	0.98702	0.98546	791	1020
4	0.99938	0.99810	553	1120
5	0.98808	0.98692	397	1200

replica more. The response time and availability of the services are given in Table 10, and one can see that the response time of service 1 has increased, but is still on a tolerable level. Another difference between Figure 9 and Figure 11 is the introduction of semi-active replicas in the latter. One can see that every service-component pair have at least one semi-active replica, except from the last component of service 5 which have neither passive nor semi-active replicas. The calculation of the number of semi-active replicas gives that the last component of service 4 has two semi-active replicas ($\lceil \frac{3-1}{1} \rceil = 2$). Lastly, it is observed that there are few passive replicas, and hence almost all replicas should be assigned enough CPU power to serve demand.

The deployment of the resulting replicas is given in Table 11, still de-

Table 11: PDC-5 *semi-active* approach: mapping between replica and node. An assignment is denoted by the amount of CPU power a replica is assigned on a node.

(i,q)	**Nodes** (CPU assignment in %)									
	1	2	3	4	5	6	7	8	9	10
(1,1)	22.8					22.8[1]				
(1,2)	11.1	11.1	11.1	11.1	11.1	0.7		11.1	11.1[1]	
(2,1)	20.0[1]	20.0					0.8		20.0	
(2,2)		6.8							6.8[1]	6.8
(2,3)		9.0[1]	9.0	9.0				9.0	9.0	
(3,1)		0.6		17.4[1]		17.4		17.4	17.4	
(3,2)	4.4		4.4	4.4		4.4[1]	4.4			
(3,3)	7.9	7.9	7.9	7.9[1]	0.7			7.9		
(4,1)				17.6			17.6[1]			
(4,2)			21.0[1]					21.0		21.0
(4,3)			25.0[1]				25.0			
(4,4)				31.7					31.7[1]	31.7[1]
(5,1)		19.0	19.0[1]		19.0				0.8	
(5,2)	33.5[1]							33.5		33.5
(5,3)						51.2[1]	51.2			
(5,4)					64.3					
Free cap	0.3	25.6	2.6	0.9	4.9	3.5	1.0	0.1	3.2	7.0

[1]Semi-active replica
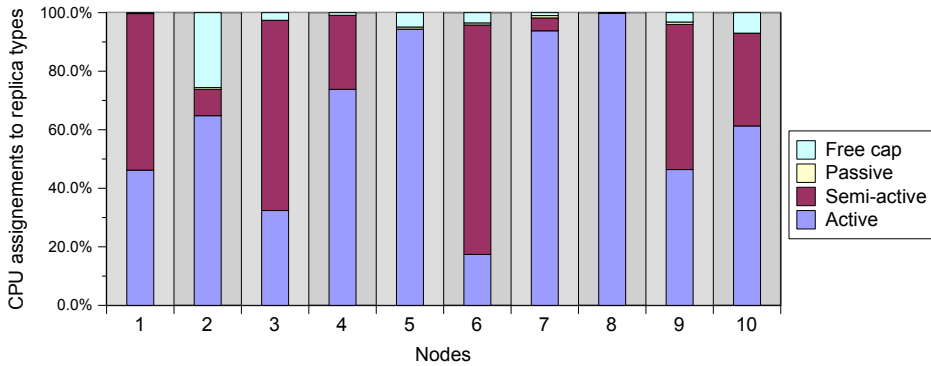
Figure 12: PDC-5 *semi-active* approach: distribution of active, semi-active and passive replicas on the nodes.

noted by the amount of CPU power assigned to the replicas. Although both active and semi-active replicas are assigned an equal amount of CPU power, the semi-active replicas are still assumed of not serving demand. In order by be distinguishable from the active replicas, the semi-active ones are marked by a footnote mark in Table 11. Compared to the capacity-lowering approach, it is now necessary to have ten nodes turned on. The reason of this increase can be explained by the fact that some passive replicas in the capacity-lowering approach now is considered semi-active. The table also shows the effect of the predeployment strategy employed in SYM2. In this test instance, the chosen active replicas to deploy is the ones of pair (1,2), and these replicas are deployed on the first five nodes. Furthermore, Figure 12 depicts the distribution of active, semi-active and passive replicas, by CPU assignment, on each node. In comparison with the same figure in the capacity-lowering approach, Figure 10, one can see that the total amount of CPU power assigned to the semi-active replicas exceeds the amount of non-assignable CPU power in the capacity-lowering approach. This makes it easier for the service provider to activate passive, or semi-active, replicas in case of failure in several active replicas, and thus reduce the probability of turning on already turned-off nodes and migrating replicas. This might imply shorter failover delay.

Since the semi-active replicas are assumed of not serving demand, their utilization equal the utilization of a corresponding passive replica. The utilization of the replicas is shown in Table 12, and one can observe that the utilization on nodes with several semi-active replicas is relatively low, cf. nodes 3 and 6 with three semi-active replicas each. The introduction of semi-active replica generally reduces the utilization compared to the capacity-lowering approach, which mainly can be explained by the increased number of turned-on nodes.

Regarding the all-active approach, the solution produced on the PDC-5 test instance is not proven to be optimal with neither strategies to reduce the number of symmetrical solutions. The results obtained by using the SYM1 strategy are presented below, and firstly, the chosen replication levels are given in Table 13. Note that the passive replicas are assumed active, and that the number of active replicas denotes the minimum num-

Table 12: PDC-5 *semi-active* approach: utilization of the nodes

(i,q)	Nodes (Utilization of the nodes in %)									
	1	2	3	4	5	6	7	8	9	10
(1,1)	16.2					0.7^1				
(1,2)	5.0	5.0	5.0	5.0	5.0	0.7		5.0	0.7^1	
(2,1)	0.8^1	11.0					0.8		11.0	
(2,2)		4.6							0.8^1	4.6
(2,3)		0.6^1	5.2	5.2				5.2	5.2	
(3,1)		0.6		0.6^1		9.7		9.7	9.7	
(3,2)	2.9		2.9	2.9		0.8^1	2.9			
(3,3)	4.6	4.6	4.6	0.7^1	0.7			4.6		
(4,1)				11.1			0.8^1			
(4,2)			0.6^1					9.2		9.2
(4,3)			1.0^1				14.6			
(4,4)				22.3					0.5^1	0.5^1
(5,1)		11.1	0.8^1		11.1				0.8	
(5,2)	1.0^1							19.1		19.1
(5,3)						0.5^1	26.6			
(5,4)					35.7					
Total	30.4	37.5	20.0	47.8	52.5	12.4	45.8	52.8	28.7	33.4

[1]Semi-active replica

ber of replicas which must be active in order to fulfill the response time requirement. The additional replicas ensure that the response time holds even in case of failures, and thus make the services satisfy the availability requirement. Since all replicas are assumed active, the decisions are only based on the total number of replicas, but the corresponding number of active and passive replicas, from the chosen replication patterns, are presented for comparison with the other approaches. By inspection, one can see that the replication levels chosen in this case equal the replication level in the capacity-lowering approach, and hence also achieve equal response time and availability values.

Table 14 presents the mapping between replicas and nodes, by showing the CPU assignment values, in the all-active approach. In this case all replicas of a service-component pair (i,q) are assigned an equal amount of CPU power and all replicas are also assumed of serving demand. The former affects the number of nodes used, and the resulting deployment configuration uses eleven nodes, compared to eight and ten in the two

Table 13: PDC-5 *all-active* approach: replication level of each component of each service

| | Number of replicas for each service-component pair (s_i, q_q) | | | | | | | | | | | | | | |
| | s_1 | | s_2 | | | s_3 | | | s_4 | | | | s_5 | | | |
	q_1	q_2	q_1	q_2	q_3	q_1	q_2	q_3	q_1	q_2	q_3	q_4	q_1	q_2	q_3	q_4
Active	2	5	2	2	4	3	4	4	1	2	1	1	2	2	1	1
Passive	1	2	2	1	1	2	1	2	1	1	1	2	2	1	1	0
Total	3	7	4	3	5	5	5	6	2	3	2	3	4	3	2	1

previous approaches. The latter makes the total utilization on the nodes more balanced, however, the utilization is still generally lower than in the capacity-lowering approach.

Table 15 gives an overview of the solutions of the three different approaches employed on the PDC-5 test instance, including the objective value, the average utilization on the turned-on nodes and the running times until the optimal solution was found. As presented the solutions are found relatively early in all cases, but in the all-active approach the objective value is not proven to be optimal within the limited maximum running time of 7000 seconds. However, the best bound is found in the root node of the B&B tree and equal the optimal objective value in the semi-active approach, 15.36, but the B&B algorithm does not manage to close the gap, by for example, proving that it is not feasible to use only ten nodes. Furthermore, as already pointed out, the average utilization of the nodes in the capacity-lowering approach is higher than the corresponding numbers in the other approaches. Considering the cost of energy usage in the three approaches, we see clearly that the capacity-lowering approach delivers the most energy-efficient solution. From the objective function, (15), we know that the number of nodes used is an important determinant, and is in fact dominating the cost. Thus, one might question if there is more reasonable to only minimize the number of nodes used. In such a case, there might be easier for the solver to close the optimality gap. Since the number of nodes naturally is an integer, any fractional lower bound can be rounded up to the nearest integer, and this might help the solver in proving the optimal solution. Lastly, the bottom row states the strategies used to reduce the number of symmetrical solutions when obtaining the solutions.

Table 14: PDC-5 *all-active* approach: mapping between replica and node. An assignment is denoted by the amount of CPU power a replica is assigned on a node. The bottom line shows the total utilization of the nodes.

(i,q)	Nodes (CPU assignment and utilization in %)										
	1	2	3	4	5	6	7	8	9	10	11
(1,1)	22.8	22.8									22.8
(1,2)		11.1	11.1			11.1	11.1		11.1	11.1	11.1
(2,1)	20.0			20.0		20.0		20.0			
(2,2)			6.8	6.8					6.8		
(2,3)		9.0		9.0	9.0		9.0		9.0		
(3,1)	17.4		17.4	17.4				17.4			17.4
(3,2)		4.4	4.4		4.4					4.4	4.4
(3,3)	7.9		7.9	7.9					7.9	7.9	7.9
(4,1)				17.6			17.6				
(4,2)						21.0	21.0	21.0			
(4,3)						25.0			25.0		
(4,4)	31.7				31.7				31.7		
(5,1)		19.0		19.0		19.0					19.0
(5,2)		33.5					33.5	33.5			
(5,3)			51.2		51.2						
(5,4)										64.3	
Free cap	0.2	0.2	1.2	2.3	3.7	3.9	7.8	8.1	8.5	12.3	17.4
Utilization	28.9	36.0	33.1	34.8	28.0	30.4	34.0	31.4	30.8	45.8	28.0

Table 15: PDC-5: summary of approaches

	All-active	Semi-active	Capacity-lowering
Used nodes	11	10	8
Avg. utilization	32.8%	36.1%	45.2%
Min cost	16.86	15.36	12.36
Solution time (s)	137[1]	1	2
Symmetry constraints	SYM1	SYM2	SYM1

[1]The objective value of the all-active approach is not proven to be optimal and has a gap between the best bound and the best solution in branch-and-bound of 8.89%. The best solution was found after 137 seconds, while max runtime was set to 7000 seconds
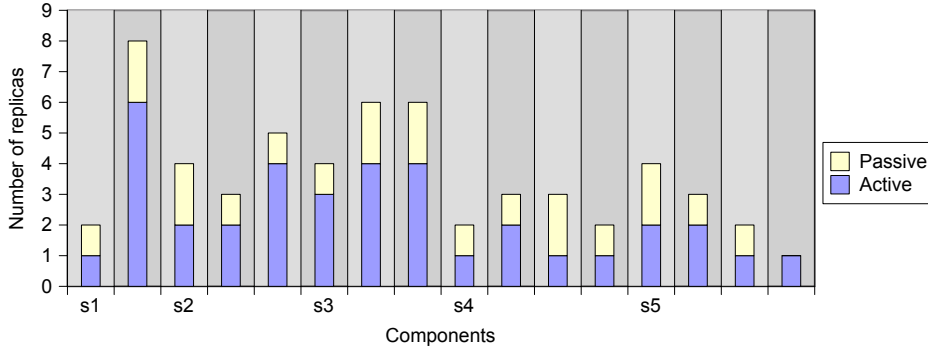
Figure 13: HCE-5 *capacity-lowering* approach: replication levels of each component of each service, denotes s1 to s5

7.3.2   The HCE-5 test instance

The services modelled in the HCE-5 test instance is the same as in the PDC-5 test instance, i.e. they have the same number of components, the same demand, the same availability and availability and response time requirements. On the other hand, the deployment environment have changed to a hybrid cloud with two public cloud domains, operated by different IaaS providers, and a single private cloud domain. Since the cost of energy usage in the private cloud is generally lower than the cost of deployment in a public cloud domain, the number of nodes is reduced to five in order to make it necessary to deploy some replicas in the public cloud.

Likewise in the presentation of the results from the PDC-5 test instance, I will firstly show the the results obtained by using the capacity-lowering approach. Figure 13 depicts the replication levels, and by inspection, we see that the total number of replicas and the number of active replicas corresponds to the semi-active approach in the SDP-PDC model. Hence the service availability and response time is shown in Table 10.

The deployment of the replicas is given in Table 16. Deployment on the nodes in the private cloud is still indicated by the amount of CPU power

Table 16: HCE-5 *capacity-lowering* approach: Deployment, CPU assignment and utilization. In the column indicating the minimum number of used domains, a value of 0 means that the component is not constrained by a such constraint. The first column from the right indicates the value of the B_{INDiq} parameter, where a value one means that the pair (i,q) have to be deployed in the private cloud (domain 1)

(i,q)	Nodes (CPU assignment and utilization in %) 1	2	3	4	5	Public cloud domains (number of replicas) 2 Act	Pas	3 Act	Pas	Min # domains used	Binding
(1,1)			22.8		0.7					0	1
(1,2)	11.1	11.1	11.1	11.1	11.1			1	2	0	0
(2,1)	20.0	20.0		0.8	0.8					0	1
(2,2)	6.8			0.8	6.8					0	1
(2,3)					0.6	4				0	0
(3,1)	17.4	17.4	17.4		0.6					1	0
(3,2)				0.8				4	1	2	0
(3,3)	7.9		7.9	0.7	0.7	2				1	0
(4,1)	17.6			0.8						0	1
(4,2)			21.0		0.6			1		2	0
(4,3)					25.0				2	1	0
(4,4)					0.5	1				1	0
(5,1)	19.0		19.0	0.8	0.8					1	0
(5,2)					1	2				1	0
(5,3)		51.2			0.5					0	1
(5,4)				64.3						0	1
Non-ass	0.0	0.0	0.0	19.2	50.7						
Free cap	0.2	0.3	0.8	0.1	0.2						
Utilization	57.2	52.4	55.8	46.0	30.4						

assigned to the replica, while deployment in the public cloud domains is given by an integer, telling the number of replicas, active and passive, that are deployed in the domain. In the HCE test instances, some components have requirements on the minimum number of domains used (cf. $S_{PREADiq}$) and other components might be bound to have all its replicas deployed in the private domain. These requirements on a component is shown in the two last columns. It is seen that the replicas of service-component pair (3,2) and (4,2) need to be deployed in at least two domains, while the other replicas have no such constraints. Some cells in the second column from the right contain a zero value, meaning that there are no constraints related to the number of used domains for the corresponding component. Note that

all pairs (i, q) with B_{INDiq} naturally have no constraints on the minimum number of domains used. In the testing of the capacity-lowering approach on the PDC-5 instance it was seen that all passive replicas were kept on the same nodes, and this characteristic is also found in Table 16, where only nodes 4 and 5 are used for deployment of passive replicas. Regarding the deployment in the public cloud domains, the results show that for a given service component, maximum one of the public domains are chosen, and furthermore no passive replicas are deployed in domain 2. The latter can partly be explained by the fact that the cost of deploying a passive replica in domain 2 is much higher than deploying the same replica in domain 3, cf. Table 5 and 6, as the amount of CPU assigned to a passive replica is less than or equal to 1 percent of the node capacity. In the case of pair (1,2), the cheapest VM type for an active replica in domain 3 is VM5 with cost 24 ¢ per hour, while if the replica instead was deployed in domain 2, the cost would have been 38 ¢ per hour. Lastly, the bottom column in Table 16 shows the utilization of the nodes, and it proves that there is a small increase in the utilization on the nodes compared to the case in the PDC-5 test instance.

Now, the focus is on the semi-active approach, and in this case the results obtained with the SYM2 strategies are presented. The solutions achieved by using the SYM2 strategy give replication levels according to Figure 14, which imply the service availabilities and response times shown in Table 17. The number of active replicas is the same as in the capacity-lowering approach (cf. Figure 13), but the total number of replicas of the components of service 4 have another distribution, affecting the availability

The deployment configuration is illustrated in Table 18, in the same

Table 17: HCE-5 *semi-active* approach: response time and availability of each service

Service	Availability	Req. availability	response time (ms)	Req. response time (ms)
1	0.99338	0.98747	557	560
2	0.99338	0.99279	613	630
3	0.99118	0.98546	791	1020
4	0.99816	0.99810	553	1120
5	0.98808	0.98692	397	1200
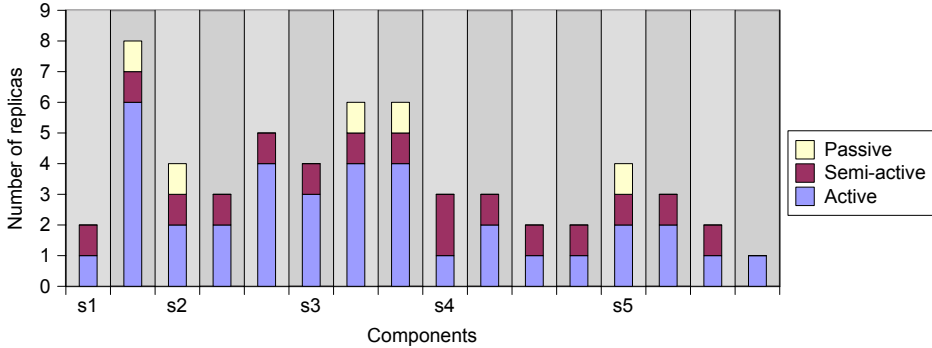
Figure 14: HCE-5 *semi-active* approach: replication level of each component of each service, denoted s1 to s5.

manner as in the capacity-lowering approach. Since the active and semi-active replicas are assigned the same amount of processing power, the semi-active replicas are again distinguished by a footnote mark. The main difference in the deployment configuration comparing with the capacity-lowering approach is that the several passive replicas now are considered semi-active, which hence lower the utilization on the nodes in the private cloud domain. This also implies that more replicas need to be deployed in the public cloud, which in turn increases the total cost of the service provider. Table 18 also shows the effect of the predeployment strategy. Among the service components bound to be deployed in the private cloud domain, (2,1) is chosen to have two active replicas predeployed on the first two nodes.

The results produced by the all-active approach, using the SYM1, strategies are quite similar to the ones that are presented above, using the semi-active approach. In fact, the replication levels are equal, although all replicas is assumed active in the case considered here. Table 19 gives an overview of the deployment of the replicas in the hybrid cloud environment, as well as the utilization on the nodes in the private cloud. Since all replicas is assumed active, a slightly higher number of replicas is deployed in the public cloud, and amplified by the fact that all replicas deployed in the public

Table 18: HCE-5 *semi-active* approach: Deployment, CPU assignment and utilization of nodes. In the column indicating the minimum number of used domains, a value of 0 means that the component is not constrained by a such constraint. The first column from the right indicates the value of the B_{INDiq} parameter, where a value one means that the pair (i, q) have to be deployed in the private cloud (domain 1)

(i,q)	Nodes (CPU assignment and utilization in %) 1	2	3	4	5	Public cloud domains (number of replicas) 2 Act[1]	Pas	3 Act[1]	Pas	Min # domains used	Binding
(1,1)		22.8		22.8[2]						0	1
(1,2)	11.1[2]	11.1	11.1	11.1	11.1			2	1	0	0
(2,1)	20.0	20.0	0.8	20.0[2]						0	1
(2,2)			6.8[2]	6.8	6.8					0	1
(2,3)						5				0	0
(3,1)				17.4[2]				3		1	0
(3,2)		0.8	4.4[2]					4		2	0
(3,3)		0.7				5				1	0
(4,1)	17.6[2]	17.6			17.6[2]					0	1
(4,2)				21.0[2]				3		2	0
(4,3)		25.0	25.0[2]							1	0
(4,4)						2				1	0
(5,1)		0.8						3		1	0
(5,2)						3				1	0
(5,3)	51.2		51.2[2]							0	1
(5,4)					64.3					0	1
Free cap	0.1	1.2	0.7	0.9	0.2						
Utilization	39.1	60.3	8.9	12.3	46.1						

[1]The column gives the number of active and semi-active replicas
[2]Semi-active replica

clouds need to be packaged in a VM instance with the CPU size corresponding to an active replica, this leads to a more expensive solution compared to the other approaches.

Table 20 sums up this section by showing the average utilization on the nodes in the private cloud domain, the usage costs in the public cloud domains, the total cost and the time it took to achieve the solutions using the Xpress solver. Comparing the average utilization of the nodes in the HCE-5 test instance against the utilization levels in the PDC-5 case, one can see that the capacity-lowering and the all-active approach has slightly higher average value, while the opposite is true for the semi-active approach. Con-

Table 19: HCE-5 *all-active* approach: deployment, CPU assignment and utilization of nodes. In the column indicating the minimum number of used domains, a value of 0 means that the component is not constrained by a such constraint. The first column from the right indicates the value of the B_{INDiq} parameter, where a value one means that the pair (i, q) have to be deployed in the private cloud (domain 1)

(i,q)	Nodes (CPU assignment and utilization in %)					Public domains (number of replicas)		Min # domains used	Bind-ing
	1	2	3	4	5	2	3		
(1,1)			22.8		22.8			0	1
(1,2)	11.1	11.1	11.1	11.1	11.1		3	0	0
(2,1)	20.0	20.0		20.0	20.0			0	1
(2,2)			6.8	6.8	6.8			0	1
(2,3)						5		0	0
(3,1)							4	1	0
(3,2)		4.4					5	2	0
(3,3)			7.9			5		1	0
(4,1)	17.6			17.6	17.6			0	1
(4,2)					21.0		2	2	0
(4,3)				25.0		1		1	0
(4,4)						2		1	0
(5,1)				19.0			3	1	0
(5,2)						3		1	0
(5,3)	51.2		51.2					0	1
(5,4)		64.3						0	1
Free cap	0.1	0.2	0.2	0.5	0.7				
Utilization	27.6	47.7	32.5	31.2	32.2				

sidering the difference in the public cloud deployment costs, we observed in the PDC-5 test instance that the capacity-lowering approach needed the fewest number of nodes to run replicas. This leads, in the case considered here, to fewer replicas deployed in the public clouds, and hence lower costs. Since the costs of deployment in public clouds are dominating the cost in the private cloud, the difference between the approaches is relatively large. This domination of the costs in the public cloud domains lead to a situation where all nodes are used in the private cloud, and hence this raises a question about the objective function. It might be argued that when considering a hybrid cloud environment, more cost factors related to the operation of the private cloud is needed if it should be worthwhile to minimize the total costs. Moreover, Table 20 also shows the time the solver used to solve the problems. In the capacity-lowering and all-active approach, the solver

needed 69 seconds and 19 second to prove optimum, respectively, but the best solutions were in fact found after only 5 seconds in both cases. When testing of the semi-active approach the solver is not able of proving that the best solution is optimal. Using the SYM2 strategy for reducing the number of symmetrical solutions the best solution is found after 16 seconds and uses the rest of the running time, until 7000 seconds, to lower the gap between the best bound and best solution to 0.64 percent. Using the SYM0 strategy in the same approach produces the same best solution, but require 321 seconds to find this solution. However, using this latter strategy, the optimality gap after 7000 seconds of running time is as low as 0.46 percent.

Table 20: HCE-5: summary of approaches

	All-active	Semi-active	Capacity-lowering
Avg. utilization of nodes	34.2%	33.3%	48.3%
Cost of using public domain 2	323.0	285.0	171.0
Cost of using public domain 3	303.0	256.5	67.5
Total cost	633.7	546.2	246.2
Solution time (s)	19	16[1]	69
Symmetry constraints	SYM1	SYM2	SYM1

[1] The objective value of the semi-active approach is not proven to be optimal and has a gap between the best bound and the best solution in branch-and-bound of 0.64%. The best solution was found after 16 seconds, while max runtime was set to 7000 seconds

7.4 Scalability of the SDP models

After showing the basic functioning of the models, the focus will now be on testing whether the models are applicable to larger problem sizes or not, and thus the results presented below do not concentrate on the details regarding deployment or replication levels. The model formulations, including both the SDP-PDC and SDP-HCE model, will be tested on the larger test instances given in Table 4, starting with the SDP-PDC model. Besides, all three approach for assignment of CPU power to passive replicas and the strategies employed to reduce the number of symmetrical solutions are included in the testing. Lastly, there will also be given comments on the scalability of the algorithm used to generate replication patterns.

7.4.1   Scalability considerations of the SDP-PDC model

The test instances include PDC-15, PDC-20 and PDC-40, which consist of 15, 20 and 40 services and 35, 40 and 80 nodes, respectively. In the PDC-15 test instance, the number of components for a given service ranges between two and five, while in the other test instances this number ranges between two and four. Firstly, each of the approaches for assignment of processing power to the passive replicas are gone through, in the same order as done above, before a comparison of the different approaches is presented. In all the larger test cases for the PDC model the maximum running time of the Xpress solver is set to 4000 seconds.

Table 21a gives an overview of the solutions produced by the capacity-lowering approach on the large test cases, together with some measures on the complexity of solving the cases. All three strategies used to reduce the number of symmetrical solutions are tested, but for all test instances, using SYM1 makes it too hard for the solver to find even a feasible solution. Thus the solver produces no solutions when employing the SYM1 constraints, but on the other hand provides reasonable good lower bounds on the objective value. Considering the PDC-15 and PDC-20 case one can observe that the usage of SYM0 provides the best solutions, but also the worst lower bound. However, considering the lower bounds produced with the help of the SYM1 and SYM2 strategies, one can acknowledge that the SYM0 strategy produces solutions with an optimality gap below 5 percent. Furthermore, it is seen that the best solutions are found within a short amount of time. Advancing to the PDC-40 case, we see a quite different situation. Now, the usage of the SYM2 strategy provides superior solutions compared to SYM0. Although, the time to find the best solution is as high as 2329 seconds. However, Figure 15a, depicting the evolution of the best solution on a time scale, shows that relatively good solutions are found within short time. For example, a solution with cost 83.54 is found after 70 seconds, while when using the SYM0 strategy, this solution is found after 2300 seconds. In Figure 15a it is also observed that the lower bound is not increased after the initial heuristics used by the Xpress solver.

Now, proceeding to the semi-active approach, the overview of the solu-

Table 21: SDP-PDC: overview of solutions and complexity on the larger test instances. In all tests on the large examples the maximum runtime of the solver is set to 4000 seconds due to the limited memory capacity. The row denoting the solution time presents the time used to find the best solution in the cases where optimum are not proved, while presents the time to prove optimum in the cases where the optimum is proved.

(a) *Capacity-lowering* approach

	PDC-15			PDC-20			PDC-40		
	SYM0	SYM1	SYM2	SYM0	SYM1	SYM2	SYM0	SYM1	SYM2
# nodes used	22		23	25		26	53		50
Avg. utilization	54.21%		51.85%	51.45%		49.96%	47.84%		50.71%
Best solution	34.19		35.69	38.79		40.29	82.04		77.54
Best bound	32.32	32.63	32.69	37.01	37.29	37.01	74.18	74.18	74.18
Gap	5.47%		8.41%	4.59%		8.14%	9.58%		4.33%
Solution time	53		37	179		66	2919		2329
# variables	3932	3950	3931	4609	4623	4609	19480	19496	19480
# constraints	3917	3963	3944	4628	4677	4662	19518	19607	19591

(b) *Semi-active* approach

	PDC-15			PDC-20			PDC-40		
	SYM0	SYM1	SYM2	SYM0	SYM1	SYM2	SYM0	SYM1	SYM2
# nodes used	31		31	35		36	72		72
Avg. utilization	38.47%		38.48%	36.80%		35.73%	35.25%		35.22%
Best solution	47.69		47.69	53.79		55.29	110.54		110.54
Best bound	46.52	47.69	47.69	53.35	53.79	53.79	108.63	108.65	109.04
Gap	2.45%		0.00%	0.82%		2.71%	1.73%		1.36%
Solution time	14		10	20		169	9		475
# variables	5711	5770	5710	6802	6862	6801	28914	29018	28914
# constraints	2085	2160	2112	2447	2531	2480	10043	10208	10116

(c) *All-active* approach

	PDC-15			PDC-20			PDC-40		
	SYM0	SYM1	SYM2	SYM0	SYM1	SYM2	SYM0	SYM1	SYM2
# nodes used	33		33	37		38	76		76
Avg. utilization	36.14%		36.14%	34.77%		33.84%	33.39%		33.37%
Best solution	50.69		50.69	56.79		58.29	116.54		116.54
Best bound	49.27	50.69	50.69	56.36	56.57	56.79	114.52	114.54	115.04
Gap	2.80%		0.00%	0.76%		2.57%	1.73%		1.29%
Solution time	1		1	3165		6	5		5
# variables	2010	2057	2010	2334	2383	2343	9817	9896	9816
# constraints	95	164	123	102	180	140	210	368	288

tions and complexity is found in Table 21b. In the PDC-15 case we can see that both usage of SYM0 and SYM2 give the same best solution, which in fact is optimal, while usage of SYM1 does not provide any solution, but the solver manages to find a lower bound on the problem that equals the optimal solution. Moreover, in the PDC-20 case, by the usage of SYM0, one is able to find the optimal solution, however the best solution is proved optimal by the best lower bounds in the case where SYM1 or SYM2 are used. Regarding the largest case, PDC-40, we observe that both usage of SYM0 and SYM2 give equal best solutions, but none of strategies can help the solver to prove optimality. Although, the SYM2 strategy provides a better lower bound, there might be argued that the SYM0 strategy achieve the best result since it is providing the same best solution as by usage of SYM2 in shorter time. Figure 15b illustrates the evolution of the best solution in the PDC-40 case using the SYM2 strategy, and the figure shows that it takes around 200 seconds to obtain a reasonable good solution, which back up the arguments above.

The results of the all-active approach are given in Table 21c, and the tests on the PDC-15 and PDC-20 cases show that the solutions produced by usage of the SYM0 strategy gives optimal solutions on both cases, when considering the lower bound produced with help of SYM2. However in the PDC-20 case, the optimal solution is found after over 3000 seconds, which might be longer than the time we are willing to wait. Figure 15c shows that by the usage of the SYM0 strategy, the solver is able to find the same solution as by employing the SYM2 strategy in a short amount of time, correctly after 3 seconds. In the largest test instance, both the usage of SYM0 and SYM2 make the solver produce the same, non-optimal solution, both within a short period of time.

Now comparing the solutions of the three approaches, one can see that the number of used nodes and the average utilization, shown in Table 21, are corresponding to the results in Table 15, in the terms of that the capacity-lowering approach is using the fewest number of nodes, and thus also has the highest utilization levels and the lowest cost. Furthermore, regarding the performance of the three different strategies to reduce the number of symmetrical solutions, one can observe that usage of the SYM0 strategy
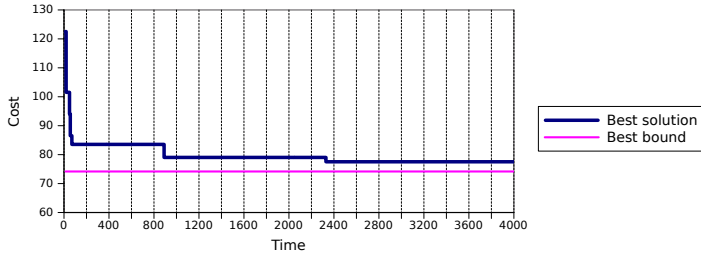
(a) *Capacity-lowering* approach: PDC-40 with SYM2
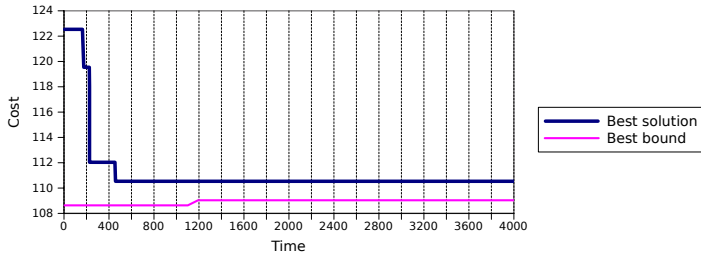
(b) *Semi-active* approach: PDC-40 with SYM2
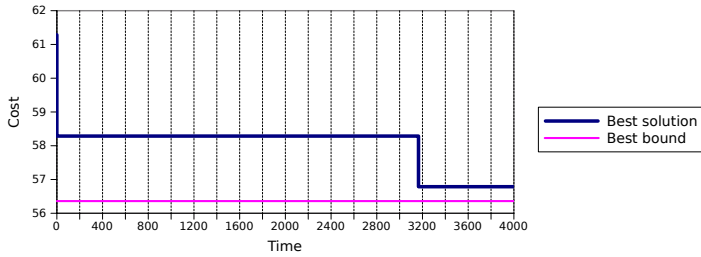
(c) *All-active* approach: PDC-20 with SYM0

Figure 15: SDP-PDC: solution graph of the large examples

always provides the best solution, with an exception in the PDC-40 case for the capacity-lowering approach. This strategy consists only of a simple predeployment of some of the replicas of a service-component pair, and one might argue that the extra constraints in the SYM1 and SYM2 strategies are unnecessary. Considering the complexity of the different approaches, we observe that the semi-active and all-active approach are solved to optimality in the PDC-15 and PDC-20 test instances, while this is not true for the capacity-lowering approach. Moreover, the capacity-lowering approach also has the highest optimality gap in the PDC-40 test instance. The greater difficulties in solving the capacity-lowering approach comparing to the other approaches, might be explained by the two bottom rows in the Tables 21a to 21c. These rows show the number of variables and constraints, after the presolve phase in the Xpress solver, of the corresponding approaches. The number of constraints in the formulation of the capacity-approach is higher than in the other two approaches, especially compared to the all-active approach. The extra number of constraints in the capacity-lowering approach, compared to the other approaches stem from the set of constraints, (29), setting the amount of non-assignable CPU power on the nodes. Although the semi-active approach has a greater number of variables, caused by the usage of the variables indicating a semi-active replica, v_{iqn}, an increase in the number of constraints often leads to higher complexity than a similar increase in the number of variables. Furthermore, the low number of variables and constraints in the all-active approach, come from the fact that neither the v_{iqn} variables nor the variables indicating an active replica, w_{iqn}, are used. Thus the constraints setting these variables, (18) and (19), and the set of constraints forcing them to be less than or equal to the deployment variable x_{iqn}, (20), are not present in the formulation. Lastly, note that it is only small differences in the number of variables and constraints between the three strategies to reduce the number of symmetrical solutions.

7.4.2   Scalability considerations of the SDP-HCE model

In the case of the SDP-HCE model, test instances, to be discussed here include the HCE-10 and the HCE-20 test instance, consisting of 10 and

20 services and 12 and 25 nodes in the private cloud domain, respectively. Likewise in the discussion of the scalability of the SDP-PDC model, all three approaches to assignment of CPU power to the passive replicas will be considered. The maximum running time of the Xpress solver is set to 7000 seconds and 4000 seconds for the HCE-10 and HCE-20 test instance, respectively.

Table 22a presents an overview of the solutions produced by the capacity-lowering approach on the HCE-10 and HCE-20 test instance, using the different strategies to reduce the number of symmetrical solutions. Likewise in the discussion of the larger PDC cases, some measures of complexity are also provided. The tests of the capacity-lowering approach, on the focused test instances, using the SYM1 strategies, give no feasible solution at all, but they provide relatively good lower bounds on the optimal solution, similar to the tests in previous subsection. Regarding the HCE-10 case, it is seen that neither using the SYM0 nor the SYM2 strategy makes the solver able to find or prove the optimal solution, but among the two strategies, SYM2 produces the solution with the lowest cost. In fact, the gap between the best found solution and the best lower bound is quite high, both when using the SYM0 strategy and when using the SYM2 strategy. However, when taking into account the best lower bound produced with the help of the SYM1 strategy, the gap between the optimal solution and the best found solution when employing SYM2 is below 25 percent. Advancing to the HCE-20 instance, it is still observed that the SYM2 strategy is superior to the other strategies, in terms of producing a low-cost solution. In this case, the optimality gap is lower, but the time used to find the best solution has increase from 180 seconds to 802 seconds. Figure 16a show the evolution of the best solution during the 4000 seconds of execution time. After only 70 seconds the solver has found a solution with an optimality gap of 6.65 percent, but regarding the best bound, the B&B procedure is not able to raise the lower bound within the time limit.

Likewise in the tests of the SDP-PDC models on the larger test instances, the semi-active approach produces solutions with lower optimality gap than the capacity-lowering approach. This is shown in Table 22b. Focusing on the HCE-10 case, usage of the SYM2 strategies produces the best

Table 22: SDP-HCE: overview of solutions and complexity on the larger test instances. In the tests on HCE-10 the maximum runtime of the solver is set to 7000 seconds, while the maximum runtime is set to 4000 in tests on HCE-20 due to the limited memory capacity. The row denoting the solution time presents the time used to find the best solution in the cases where optimum are not proved, while presents the time to prove optimum in the cases where the optimum is proved.

(a) *Capacity lowering* approach

	HCE-10			HCE-20		
	SYM0	SYM1	SYM2	SYM0	SYM1	SYM2
Best solution	207.11		195.11	3084.41		2899.94
Best bound	133.14	146.58	133.14	2736.28	2738.25	2736.52
Gap	35.72%		31.76%	11.29%		5.64%
Solution time	1203		180	641		802
# variables	940	948	938	3775	3812	3775
# constraints	1059	1064	1058	4168	4224	4186

(b) *Semi-active* approach

	HCE-10			HCE-20		
	SYM0	SYM1	SYM2	SYM0	SYM1	SYM2
Best solution	823.40		818.91	4805.69		4776.20
Best bound	805.11	805.92	805.92	4725.45	4724.58	4725.09
Gap	2.22%		1.59%	1.67%		1.07%
Solution time	3622		6041	2178		2030
# variables	1341	1366	1339	5457	5524	5455
# constraints	707	728	711	2609	2664	2625

(c) *All-active* approach

	HCE-10			HCE-20		
	SYM0	SYM1	SYM2	SYM0	SYM1	SYM2
Best solution	942.40	1098.88	956.90	5678.62		5682.62
Best bound	934.85	934.85	934.95	5593.39	5593.76	5593.39
Gap	0.80%	14.93%	2.29%	1.50%		1.57%
Solution time	4576	6197	4	1		2
# variables	479	494	477	1925	1962	1923
# constraints	151	167	155	526	575	542

solutions, whereas the usage of SYM1 makes it too difficult for the solver to find a feasible solution. Although Table 22b indicates that the best solution is found after more than 6000 seconds, Figure 16b shows that a good solutions are produced early, specifically a solution with an optimality gap of 3.20 percent is found after only 5 seconds. Proceeding to the HCE-20 test instance, still the usage of SYM2 gives the best solution. Although the solution is not optimal, comparing with the HCE-10 case, we observe that the optimality gap is slightly smaller. Figure 16c reveals that even thought the B&B algorithm in the Xpress solver finds the best solution after over 2000 seconds, relatively good solutions are found within the first seconds of the execution, in this case as well.

The results of the all-active approach, given in Table 22c, show that the by the usage of the SYM1 strategy the solver is in fact capable of providing a feasible solution in the HCE-10 test instance. However, with the help of the SYM0 strategy, the solver is providing a near-optimal solution after 3600 seconds. Again, it is shown, in Figure 16d, that good solutions are also found early in the search. The usage of the SYM2 strategy produces a relatively good solution after only 4 seconds, but in fact, using the SYM0 strategy produces an even better solution within the same amount of time. In the largest HCE test instance, both the usage of the SYM0 and the SYM2 strategy produces near-optimal solutions during the first seconds of execution.

By comparing the three approaches, and regarding the performance of the strategies used to reduce the number of symmetrical solutions, one can see from Table 22 that the usage of the SYM0 strategy in the all-active approach gives the best solutions in both the HCE-10 and the HCE-20 case, while the usage of SYM2 is superior to SYM0 in the other two approaches. However, the lower bounds produced by the usage of the two strategies are almost the same. It is although noticeable that the SYM2 strategy makes the solver finding good solution in shorter or the same amount of time, compared to SYM0, since it uses a set of constraints to sort the nodes in the private cloud domain, according to whether they are turned off or not, while indeed, all nodes are used and turned on. As a matter of fact, the constraints in SYM2 are removed in the presolve phase of the Xpress solver.
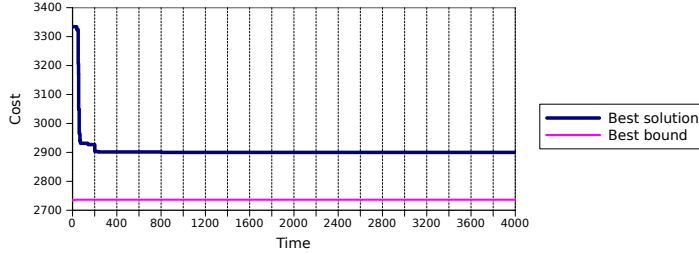
(a) *Capacity-lowering* approach: HCE-20 with SYM2
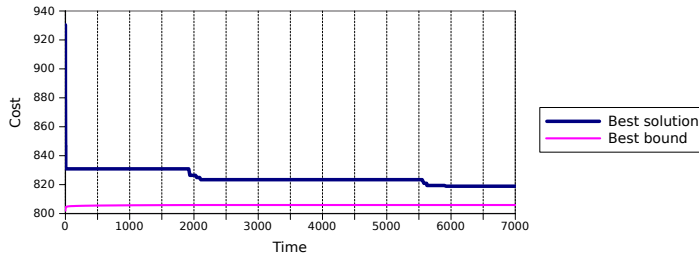
(b) *Semi-active* approach: HCE-10 with SYM2
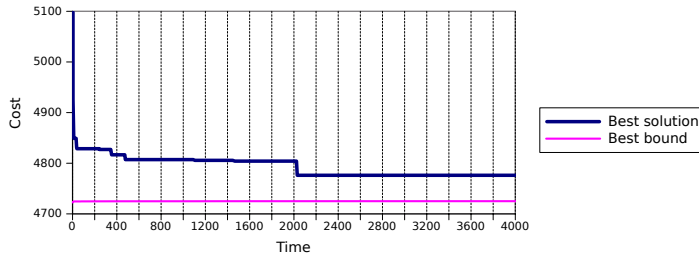
(c) *Semi-active* approach: HCE-20 with SYM2
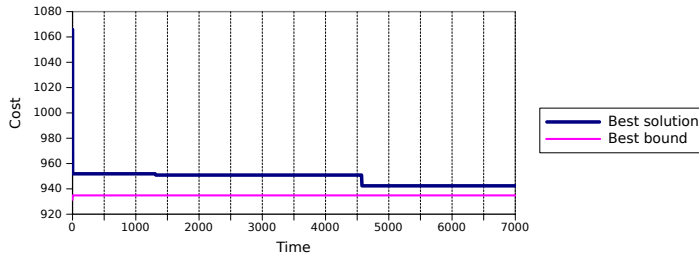
(d) *All-active* approach: HCE-10 with SYM0

Figure 16: SDP-HCE: solution graph of the large examples

Considering the complexity in solving the test instances, it is observed that none test instances is solved to optimality, but the models of the semi-active and all-active approach give near-optimal solutions in both test instances. Likewise from the results of the larger PDC test instances, one can observe that the capacity-lowering approach has the highest number of constraints compared to the other two approaches, which might explain why this approach is the most complex to solve, in terms of the optimality gap. Comparing with the PDC examples, we also see that all approaches have a higher number of constraints relative to the number of variables. This is caused by the set of constraints, (52), setting the fraction of demand which is handled from the private cloud domain. This set of constraints is done for each replication pattern of each service, and hence, since the number of replication patterns can be large in some cases, this set of constraints might seriously affect the scalability of the model. The replication pattern generation will be commented next.

7.4.3 Scalability of the replication pattern generation

If one does not take any measures to limit the number of replication patterns, this number might be enormous. In principle, the number of replication patterns of a service grow exponentially with the number of components constituting the service. Assume that one is to enumerate all possible combinations of the number of active and passive replicas for single component. With the limit that the total number of replicas of this component is below ten, the total number of combinations is $10 \cdot \frac{10+1}{2} = 55$. Thus if a service consists of $\mathbf{Q}$ such components, the total number of replication patterns for this service is given by $55^{\mathbf{Q}}$. Even supposing that we eliminate the replication patterns which not satisfy the response time and availability requirements, we might still get a very high number of replication patterns, making the proposed MILP models difficult to solve. In order to limit the cardinality of the set of replication patterns even further, without affecting the optimal solution, I have implemented the two additional tests, shown in Algorithm 6.1 and described in Section 6.5. These tests seem to work fairly well, and the average number of replication patterns for services with

two, three, four and five components is given in Table 23, where the average values are calculated over the events in the PDC test instances. The first data row in the table shows the total number of services that have the different number of components, over all PDC cases. The PDC-15 is the only test instance that contains services with five components, and hence the corresponding average value is based on a lower number of events. Although, Table 23 shows that the average number of replication patterns of a service is far lower than the theoretical values, this average tends to be increasing exponentially with the number of components of the service, and thus might be regarded as a limitation of the model. Besides, a simple test on the PDC instances shows that an increase in the number of components have larger effect on the time used by the algorithm to generate the replication patterns than an increase in the number of services. It is observed that the time to generate the replication patterns in the PDC-20 case and PDC-40 case is approximately 2 seconds and 8 seconds, while in the PDC-15 case, where three services have five components, the time spent is approximately 45 seconds. This can be explained by that, even though the generation algorithm does not updates the sets , $\mathcal{R}_i$, with patterns not satisfying the SLA requirements, the loops are still iterating all possible combinations. This must be said to be the most prominent deficiency of the algorithm, and thus if the algorithm should be tested on services with a larger number of components, there is a need for developing new stopping criteria in the loops.

Table 23: The average number of replication patterns for services with 2,3,4 and 5 components in test instances PDC-5, PDC-15, PDC-20 and PDC-40

	Number of components			
	2	3	4	5
# services	26	33	28	3
Avg. # replication patterns	1.3	1.4	4.1	19.0

7.5 Performance and dependability concerns

Adequacy of the approaches for CPU assignment to passive replicas

In the development of the models it became apparent that several modelling approaches could to be taken regarding the CPU assignment to passive replicas. In the previous sections three approaches related to this have been tested, and among them the capacity-lowering approach have shown to give the highest utilization of the nodes. In the PDC case this means that more nodes are turned off, whereas, in the HCE case, fewer replicas need to be deployed in the public cloud. However in this consideration, a question that arises relates to the dependability aspects of the various approaches. A failure in one replica is assumed of not having an impact on the probability of failure in other replicas. Are the three approaches affecting this assumption? Yes, the capacity-lowering approach and semi-active approach can be said to conflict with this assumption since it is not completely guaranteed that the standby replicas have access to enough CPU resources to be able to serve demand in case of failures in more than one active replicas. In this case, the standby replicas might need to be migrated to another node or into the public cloud, and a migration of a replica might take longer time than is acceptable in terms of the service quality, specified in the SLAs. As dependability and performance are tightly coupled, long failover delays due to migration might not only lead to a less available service, but also higher response times. Thus it can be stated that the capacity-lowering approach and the semi-active approach trade better power efficiency for less performance.

Adequacy of the availability model

A second question that emerge is whether the assumption about no failure interdependency among the replicas is valid or not. Even though the virtualization technology promises to isolate the software faults in a VM from the other VMs on the same physical machine, failures in a physical node or in the management system of a data center or cloud potentially affects all

VMs associated with this node or data center. These failures are omitted in the reliability block model, used to quantify the availability of the services, and are instead covered in the MILP model through requirements for node-disjoint and domain-disjoint deployment. By choosing to model the problem in this manner some, aspects of the dependability is lost, and thus it would be interesting to investigate what effect this choice of modelling have on the availability observed from the end-users' viewpoint.

Adequacy of the response time model

The focused performance metric in the proposed models is the response time of the services, and in this sense, I have neglected the network delay and rather only considered the sojourn time in the service components. This is clearly a weakness of the models since the network delay might be of importance in services requiring low response time. Solomianik (2009) presents empirical values of the network latency inside and between Amazon EC2 Availability Zones[3], and the results show that the average round trip times between VM instances inside an Availability Zone might be as high as 400ms, whereas between Availability Zones this average is approximately 1000ms. For services requiring low latency and response time, these concerns need to be considered in the deployment decisions, and the latter latency component might confine deployment of collaborating components in the same domain.

---

[3]Amazon denotes the different geographical locations of their data centres as Availability Zones. This corresponds to the different public domains in my models

8   Conclusions

I have presented two main MILP models designed to support service providers in their decisions related to provisioning of their distributed services. The focused decisions regarded the choice of replication levels for the services, in order to achieve the required QoS, and the deployment of the resulting replicated software components. In the first MILP model, the deployment decisions were limited to the virtualized private data center of the service provider, while the second MILP expanded this environment to include the opportunities of Cloud computing, by letting the service provider have the option deploy the replicated components in a public cloud. Hence, the latter deployment problem was modelled in a hybrid cloud environment.

The objective in the SDP-PDC model was to promote an energy-efficient deployment architecture, and thus the costs of energy usage was sought minimized, simultaneously as the service provider were to deliver a service in accordance with the SLA requirements. Energy-efficient deployment was mainly achieved by consolidating the software components on the smallest subset of nodes, bounded by the nodes' CPU capacity, and thus be able to turn off unused nodes. There was questioned if it would have been more reasonable to simplify the current objective function to only be minimizing the number of turned-on nodes since this number was the determining factor in the energy usage.

With the advent of the public clouds in the SDP-HCE model, the objective was extended to include the cost of deploying the services in these clouds. Consequently, the total costs were sought minimized, while still ensuring that the services satisfied the QoS requirements. Since the cost of deployment in the public clouds dominated the cost of energy usage, there was argued that in order to make it useful to minimize the total cost, more operational costs of the private cloud need to be taken into account.

The focused QoS parameters were response time and availability, and queueing theory and reliability block diagrams were used to relate the replication level decisions with these parameters. If directly included in the optimization model, the usage of these methods would have given a nonlinear programming model, so instead an algorithm, generating feasible

replication patterns of a service, was constructed. A replication pattern of a given service consists of a number of active and passive replicas for each component of the service, and due to the complex nature of dependability modelling, the number of SLA-satisfying replication patterns grow exponentially with the number of components. Although there have been implemented successful measures to limit the number of generated replication patterns, some simple tests showed that the time spent on this generation increases severely with the number of components.

It is important to guarantee that a passive replica needs short time to become active and serve demand in case of a failure in one of the active replicas, and in order to ensure this, three different approaches for assignment of CPU resources to the passive replicas were tested. Firstly, the *all-active* approach considered all replicas as active, and hence all replicas were assigned enough CPU power to serve demand. Secondly, a more resource friendly approach were taken, where only a subset of the passive replicas were assigned the same amount of CPU power as the active replicas. These replicas were termed semi-active, and thereof the approach was named the *semi-active* approach. The last, approach considered to lower the assignable CPU power on a node in order to let at least one passive replica deployed on this node be able to turn active. It was seen that the capacity-lowering approach achieved the best results in terms of energy efficiency, but simultaneously it was argued that this approach challenged the assumption about no interdependence in replica failures. On the other hand, the all-active approach consumed the most energy, and was also said to agree with the assumption. These considerations illustrate the trade-off between energy efficiency and performance.

The MILP models were tested on test instances of varying size, and there was seen that when the problem scale grew, the optimal solution could not always be guaranteed. Since the nodes was assumed homogeneous, there were implemented strategies to remove symmetrical solutions, and the results of the SDP-PDC showed that a simple strategy, based on predeployment of some replicas, gave good solutions, some optimal, in a short amount of time. On the other hand, the results of the SDP-HCE depicted that this problem was harder to solve, but however, the optimality

gaps in the semi-active and all-active approach were below 2 percent. The reason behind the higher gaps in the HCE model, compared the PDC model, was argued to stem from the higher number of constraints. Furthermore, there was seen that the capacity-lowering approach was generally harder to solve compared to the other approaches, which, also in this case, was caused by the higher number of constraints in the former.

Lastly, it was illustrated that, in the cases where an optimal solution was not found, the solver found good solutions during the first seconds of execution. In tactical planning problems, like the ones that are presented in this thesis, it might be of greater importance to find good solutions quickly, rather than waiting a long time for obtaining the optimal solution. With this in mind, the proposed models can be said to perform well on the provided test instances.

9 Future work

In this last section some possible directions in the future work and extension of the current models will be outlined. Some of the aspects mentioned below has already received a notice earlier in this thesis, like introduction of communication costs in the cost model of the public clouds and consideration of network delay.

Communication

The current models acknowledge that a service consists of a set of collaboration software components, but, apart from that, the generation of replication patterns considers these components as chains of servers, the collaborations are not modelled explicitly. As identified earlier, the providers of a public IaaS cloud charges a significant cost per gigabyte of data transferred in and out of the VM instances, and furthermore, it is stated that the intradomain and interdomain network latency are substantial. These aspects are sought included in further models. The latter raise a question about whether the network delay should be included directly in the response time calculation of the algorithm or if the delay should be modelled as a explicit set of constraints in the MILP model. Moreover, in a real system, load-balancers are used to split the demand between several active replicas, and one must consider if it is expedient to include the modelling of these.

Generation of replication patterns

The performance tests done on the algorithm generating the replication patterns have been few and simple, but they show that, even though the number of generated replication patterns are limited compared to the theoretical values, the time it spends on the generation grow sharply with the number of components. In order to improve the performance of the algorithm, it is necessary to test it more thoroughly. However, at this moment of writing, developing new stopping criteria might be considered as a means to increase the performance.

Availability analysis

In this thesis, I have implemented three approaches for assignment of CPU power to passive replicas, and as already elaborated, one can question the observed response time and availability in the capacity-lowering and the semi-active approach. Hence, future work might include developing a simulation model used to examine the effects these approaches have on the service quality. Besides, the node failures, link failures and failures in the management system of a data center or cloud are handled through explicit constraints ensuring node-disjoint deployment and, to a certain degree, domain-disjoint deployment. The effects of these constraints, especially the latter, would be interesting to investigate in future work.

Test instances

The proposed models have been tested on more or less realistic data, but in order to achieve realistic solutions the models need to be tested on larger and more realistic test instances. To incorporate even more realism in the parameter settings, there might be necessary to study Internet services and cloud systems in detail, including, but not restricted to demand, capacities and availability and response time of components. Hence it is essential to include a such study in future work.

Time periods

In order to introduce more dynamics in the models it could be possible to expand the models to include two or more time periods. As stated in Section 3, there is a certain switching cost of turning servers on and off, in terms of delays in turning the servers on and additional power draw. Hence, a periodic model can take into account these switching costs and also, for example, varying demand in the periods. In such a way, the replication levels might change between periods and migration of the virtual machines holding the replicas might be modelled. A multi-period model can also form the basis of a stochastic programming model.

Developing heuristic methods

Several articles in the literature, related to the problems modelled herein, report difficulties of solving large test cases, and hence promote the need for a heuristic approach. At the time of writing, the focus will not be on developing heuristic methods, but in a longer time horizon, doing this might be relevant.

References

Amazon EC2 website, 2011. Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`. Last visited 2011-05-25.

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andy Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, 2010.

Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. Fundamental Concepts of Dependability, 2001.

Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, January 2004. ISSN 1545-5971. doi: 10.1109/TDSC.2004.2.

Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.

Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Y. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, 82:47–111, 2011.

Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(3): 299 –316, june 2000. ISSN 1063-8210. doi: 10.1109/92.845896.

Richard Brown, Eric Masanet, Bruce Nordman, Bill Tschudi, Arman Shehabi, John Stanley, Jonathan Koomey, Dale Sartor, Peter Chan, Joe Loper, Steve Capana, Bruce Hedman, Rebecca Duff, Evan Haines, Danielle Sass, and Andrew Fanara. Report to congress on server and data center energy efficiency: Public law 109-431. Technical report, Ernest Orlando Lawrence Berkeley National Laboratory, University of California, 2008.

U.S. Energy Information Administration EIA. Average retail price of electricity to ultimate customers by end-use sector, 2011. URL `http://www.eia.gov/cneaf/electricity/epm/table5_6_a.html`. Last visited 2011-04-30.

Peder Emstad, Poul Heegaard, Bjarne Helvik, and Laurent Paquerau. *Dependability and performance in information and communication systems - Fundamentals*. Tapir academic publisher, 2008. Compendium in subject Dependability and Performance with Discrete Event Simulation at NTNU.

Xiaobo Fan, Wolf-Dietrich Weber, and Luiz André Barroso. Power provisioning for a warehouse-sized computer. In Dean M. Tullsen and Brad Calder, editors, *ISCA*, pages 13–23. ACM, 2007. ISBN 978-1-59593-706-3.

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

GoGrid Cloud hosting website, 2011. GoGrid Cloud hosting. http://www.gogrid.com/cloud-hosting/, Last visited 2011-05-25.

Anders N. Gullhav. Service deployment problems with replicated and non-replicated components, Dec 2010. Specialization project at IØT, NTNU.

Bjarne Helvik. *Dependable Computing Systems and Communication Networks; Design and Evaluation*. Tapir academic publisher, 2007. Draft lecture notes.

Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia L. Lawall. Entropy: a consolidation manager for clusters. In Antony L. Hosking, David F. Bacon, and Orran Krieger, editors, *VEE*, pages 41–50. ACM, 2009. ISBN 978-1-60558-375-4.

ITU-T. Terms and definitions related to quality of service and network performance including dependability, Aug 1994. ITU-T Recommandation E.800.

Loir M. Kaufman. Data security in the world of cloud computing. *Security Privacy, IEEE*, 7(4):61 –64, july-aug. 2009. doi: 10.1109/MSP.2009.87.

Jean-Claude Laprie, Algirdas Avizienis, and Hermann Kopetz, editors. *Dependability: Basic Concepts and Terminology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992. ISBN 0387822968.

Peter Mell and Tim Grance. The NIST definition of Cloud Computing, version 15. `http://csrc.nist.gov/groups/SNS/cloud-computing/`, Jul 2009. Last visited 2010-12-08.

Daniel A. Menasce and Virgilio A. F. Almeida. *Capacity Planning for Web Services: metrics, models, and methods*. Prentice Hall, 2001.

Lauri Minas and Brad Ellison. *Energy efficiency for information technology: How to reduce power consumption in servers and data centers*. Intel Press, Aug 2009. Second-hand referrence.

Vinicius Petrucci, Orlando Loques, and Daniel Mossé. A dynamic optimization model for power and performance management of virtualized clusters. In Herman de Meer, Suresh Singh, and Torsten Braun, editors, *e-Energy*, pages 225–233. ACM, 2010. ISBN 978-1-4503-0042-1.

Rackspace Cloud hosting website, 2011. Rackspace Cloud hosting products. `http://www.rackspace.com/cloud/cloud_hosting_products/`. Last visited 2011-05-25.

Lei Rao, Xue Liu, Marija Ilic, and Jie 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*, ICCPS '10, pages 188–197, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0066-7. doi: http://doi.acm.org/10.1145/1795194.1795220.

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

Oren Solomianik. Network latency inside and across Amazon EC2 Availability Zones, May 2009. URL `http://orensol.com/2009/05/24/network-latency-inside-and-across-amazon-ec2-availability-zones/`. Website last visited 2011-05-10.

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

Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 228 –235, july 2010. doi: 10.1109/CLOUD.2010.58.

Jeffrey M. Voas and Jia Zhang. Cloud computing: New wine or just a new bottle? *IT Professional*, 11(2):15–17, 2009.

Lizhe Wang, Jie Tao, Marcel Kunze, Alvaro Canales Castellanos, David Kramer, and Wolfgang Karl. Scientific cloud computing: Early definition and experience. In *HPCC*, pages 825–830. IEEE, 2008. ISBN 978-0-7695-3352-0.

Simon Wardley, Etienne Goyer, and Nick Barcet. Ubuntu Enterprise Cloud architecture. Technical report, Canonical, Aug 2009.

Kaiqi Xiong and Harry G. Perros. Computer resource optimization for differentiated customer services. In *MASCOTS*, pages 226–238. IEEE Computer Society, 2006. ISBN 0-7695-2573-3.

Moshe Zukerman. Introduction to queueing theory and stochastic teletraffic models. `http://www.ee.cityu.edu.hk/~zukerman/classnotes.pdf`, 2010.

A   Summary of the MILP models

A.1   SDP-PDC formulation

Sets

$\mathcal{S}$ Set of all services, indexed by i

$\mathcal{Q}_i$ Set of components of service i, indexed by q

$\mathcal{R}_i$ Set of replication patterns for service i, indexed by r

$\mathcal{N}$ Set of nodes, indexed by n

Parameters

H_{RS}	The length of the time period, in hours
C_{OSTPWR}	Cost per unit of energy used
$P_{WRCOEFF}$	Coefficient translating CPU utilization to power usage
P_{WRIDLE}	The power consumption of an idle node
D_{EMi}	The avg. number of request for service i per time unit
$J_{OBLOADiq}$	The avg. CPU power needed to handle a request in (i, q) per time unit
$C_{PUDEMiq}$	The CPU power demanded by service requests for (i, q) $= D_{EMi}J_{OBLOADiq}$
$C_{PUASSiq}$	The amount of CPU power assigned to an active replica of (i, q)
V_{ARi}	The parameter ensuring that a service can handle the peaks in demand
C_{PUOHiq}	CPU power overhead for management-related tasks in replicas
C_{APCPU}	CPU capacity on node
N_{Riqr}	The total number of replicas of (i, q) using replication pattern r
A_{CTRiqr}	The number of active replicas of (i, q) using replication pattern r
$S_{ACOEFFiq}$	Coefficient used to control the number of semi-active replicas of (i, q)

Varibles

$$x_{iqn} \quad = \begin{cases} 1 & \text{if the a replica of } (i, q) \text{ is deployed on node } n \\ 0 & \text{otherwise} \end{cases}$$

$$w_{iqn} \quad = \begin{cases} 1 & \text{if a replica of } (i, q) \text{ deployed on node } n \text{ is active} \\ 0 & \text{otherwise} \end{cases}$$

$$v_{iqn} \quad = \begin{cases} 1 & \text{if a replica of } (i, q) \text{ deployed on node } n \text{ is semi-active} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ir} \quad = \begin{cases} 1 & \text{if replication pattern } r \text{ is used for service } i \\ 0 & \text{otherwise} \end{cases}$$

$$o_n \quad = \begin{cases} 1 & \text{if node } n \text{ is turned on} \\ 0 & \text{otherwise} \end{cases}$$

m_n The amount of non-assignable CPU power on node n in the capacity-lowering approach

Objective function

$$\begin{aligned} \min z_{PDC} = C_{OSTPWR} H_{RS} \Big(& \sum_{n \in \mathcal{N}} P_{WRIDLE} o_n \\ & + P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{n \in \mathcal{N}} C_{PUOHiq} x_{iqn} \\ & + P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUDEMiq} \Big) \end{aligned} \tag{A.1}$$

A.1.1   Constraints in the all-active approach

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1, \quad \forall i \in \mathcal{S} \tag{A.2}$$

$$\sum_{n \in \mathcal{N}} x_{iqn} - \sum_{r \in \mathcal{R}_i} N_{Riqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{A.3}$$

$$\sum_{i\mathcal{S}} \sum_{q \in \mathcal{Q}_i} (C_{PUASSiq} V_{ARi} + C_{PUOHiq}) x_{iqn} - C_{APCPU} o_n \leq 0, \quad \forall n \in \mathcal{N} \tag{A.4}$$

$$x_{iqn} \in \{0,1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{A.5}$$

$$y_{ir} \in \{0,1\}, \quad \forall i \in \mathcal{S}, r \in \mathcal{R}_i \tag{A.6}$$

$$o_n \in \{0,1\}, \quad \forall n \in \mathcal{N} \tag{A.7}$$

A.1.2   Constraints in the semi-active approach

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1, \quad \forall i \in \mathcal{S} \tag{A.8}$$

$$\sum_{n \in \mathcal{N}} x_{iqn} - \sum_{r \in \mathcal{R}_i} N_{Riqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{A.9}$$

$$\sum_{n \in \mathcal{N}} w_{iqn} - \sum_{r \in \mathcal{R}_i} A_{CTRiqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{A.10}$$

$$\sum_{n \in \mathcal{N}} v_{iqn} - \sum_{r \in \mathcal{R}_i} \left\lceil S_{ACOEFFiq} \frac{N_{Riqr} - A_{CTRiqr}}{A_{CTRiqr}} \right\rceil y_{ir} = 0, \ \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{A.11}$$

$$w_{iqn} + v_{iqn} - x_{iqn} \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{A.12}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUASSiq} V_{ARi} w_{iqn} + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUASSiq} V_{ARi} v_{iqn} +$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUOHiq} x_{iqn} - C_{APCPU} o_n \leq 0, \quad \forall n \in \mathcal{N} \tag{A.13}$$

$$x_{iqn} \in \{0,1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{A.14}$$

$$w_{iqn} \in \{0,1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{A.15}$$

$$v_{iqn} \in \{0,1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{A.16}$$

$$y_{ir} \in \{0,1\}, \quad \forall i \in \mathcal{S}, r \in \mathcal{R}_i \tag{A.17}$$

$$o_n \in \{0,1\}, \quad \forall n \in \mathcal{N} \tag{A.18}$$

A.1.3   Constraints in the capacity-lowering approach

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1, \quad \forall i \in \mathcal{S} \tag{A.19}$$

$$\sum_{n \in \mathcal{N}} x_{iqn} - \sum_{r \in \mathcal{R}_i} N_{Riqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{A.20}$$

$$\sum_{n \in \mathcal{N}} w_{iqn} - \sum_{r \in \mathcal{R}_i} A_{CTRiqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{A.21}$$

$$w_{iqn} - x_{iqn} \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{A.22}$$

$$m_n - C_{PUASSiq} V_{ARi}(x_{iqn} - w_{iqn}) \geq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{A.23}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUASSiq} V_{ARi} w_{iqn} +$$
$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUOHiq} x_{iqn} + m_n - C_{APCPU} o_n \leq 0, \quad \forall n \in \mathcal{N} \tag{A.24}$$

$$x_{iqn} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{A.25}$$

$$w_{iqn} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, n \in \mathcal{N} \tag{A.26}$$

$$y_{ir} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, r \in \mathcal{R}_i \tag{A.27}$$

$$o_n \in \{0, 1\}, \quad \forall n \in \mathcal{N} \tag{A.28}$$

A.2   SDP-HCE formulation

Sets

$\mathcal{S}$	Set of all services, indexed by i
$\mathcal{Q}_i$	Set of components of service i, indexed by q
$\mathcal{R}_i$	Set of replication patterns for service i, indexed by r
$\mathcal{D}$	Set of cloud domains for deployment, indexed by d
$\mathcal{D}_{PR}$	Set of private cloud domains for deployment, indexed by d
$\mathcal{D}_{PU}$	Set of public cloud domains for deployment, indexed by d
$\mathcal{N}_d$	Set of nodes for deployment in a private cloud domain d, indexed by n

Parameters

H_{RS}	The length of the time period, in hours
C_{OSTPWR}	Cost per unit of energy usage
$P_{WRCOEFF}$	Coefficient translating CPU utilization to power usage
P_{WRIDLE}	The power consumption of a node in idle state
D_{EMi}	The avg. number of request for service i per time unit
$J_{OBLOADiq}$	The avg. CPU power needed to handle a request in (i, q) per time unit
$C_{PUDEMiq}$	The CPU power demanded by service requests for (i, q) $= D_{EMi} J_{OBLOADiq}$
$C_{PUASSiq}$	The amount of CPU power assigned to an active replica of (i, q)
V_{ARi}	The parameter ensuring that a service can handle the peaks in demand
C_{PUOHiq}	CPU power overhead for management-related tasks in replicas
C_{APCPU}	CPU capacity on each node in a private cloud domain

N_{Riqr}	The total number of replicas of (i,q) using replication pattern r
N_{RMXiq}	The maximum of N_{Riqr} over r
A_{CTRiqr}	The number of active replicas of (i,q) using replication pattern r
$A_{CTRMXiq}$	The maximum of A_{CTRiqr} over r
$S_{ACOEFFiq}$	Coefficient used to control the number of semi-active replicas of (i,q)
$C_{OSTVMACTiqd}$	Cost of using a VM instance for the active replicas of (i,q) in public cloud domain d per hour
$C_{OSTVMPASiqd}$	Cost of using a VM instance for the passive replicas of (i,q) in public cloud domain d per hour
$S_{PREADiq}$	The required number of cloud domains used for a replica in percentage of the total number of replicas
B_{INDiq}	Equal d if pair (i,q) only can be deployed in cloud domain d, and equal 0 if there is no binding on the domains used for deployment.

Variables

$$
u_{iqd} = \begin{cases} 1 & \text{if the a replica of } (i,q) \text{ is deployed in cloud domain } d \\ 0 & \text{otherwise} \end{cases}
$$

x_{Diqd} The total number of replicas of the pair (i,q) that is deployed in cloud domain d

$$
x_{iqdn} = \begin{cases} 1 & \text{if the a replica of } (i,q) \text{ is deployed on node } n \text{ in private cloud domain } d \\ 0 & \text{otherwise} \end{cases}
$$

w_{Diqd} The number of active replicas of (i,q) that is deployed in cloud domain d

$$
w_{iqdn} = \begin{cases} 1 & \text{if an active replica of } (i,q) \text{ is deployed on node } n \text{ in private cloud domain } d \\ 0 & \text{otherwise} \end{cases}
$$

v_{Diqd} The number of semi-active replicas of (i, q) deployed in cloud domain d

v_{iqdn} $= \begin{cases} 1 & \text{if a semi-active replica of } (i,q) \text{ is deployed on node } n \text{ in} \\ & \text{private cloud domain } d \\ 0 & \text{otherwise} \end{cases}$

y_{ir} $= \begin{cases} 1 & \text{if replication pattern } r \text{ is used for service } i \\ 0 & \text{otherwise} \end{cases}$

o_{dn} $= \begin{cases} 1 & \text{if node } n \text{ in private cloud domain } d \text{ is turned on} \\ 0 & \text{otherwise} \end{cases}$

m_{dn} The amount of non-assignable CPU power on node n in private cloud domain $d \in \mathcal{D}_{PR}$, in the capacity-lowering approach

f_{iqd} The fraction of demand-service replicas of (i, q) that is deployed in private cloud domain d

A.2.1   All-active approach

Objective function

$$
\begin{aligned}
z_{HCEAA} = C_{OSTPWR} H_{RS} \Bigg(& \sum_{d \in \mathcal{D}_{PR}} \sum_{n \in \mathcal{N}_d} P_{WRIDLE} o_{dn} \\
+ P_{WRCOEFF} & \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PR}} \sum_{n \in \mathcal{N}_d} C_{PUOHiq} x_{iqdn} \\
+ P_{WRCOEFF} & \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PR}} C_{PUDEMiq} f_{iqd} \Bigg) \\
+ H_{RS} & \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PU}} C_{OSTVMACTiqd} x_{Diqd}
\end{aligned}
\tag{A.29}
$$

Constraints

$$
\sum_{r \in \mathcal{R}_i} y_{ir} = 1, \quad \forall i \in \mathcal{S}
\tag{A.30}
$$

$$
\sum_{d \in \mathcal{D}} x_{Diqd} - \sum_{r \in \mathcal{R}_i} N_{Riqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i
\tag{A.31}
$$

$$\sum_{n \in \mathcal{N}_d} x_{iqdn} - x_{Diqd} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR} \tag{A.32}$$

$$\sum_{d \in \mathcal{D}} u_{iqd} - \sum_{r \in \mathcal{R}_i} N_{Riqr} S_{PREADiq} y_{ir} \geq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{A.33}$$

$$x_{Diqd} - u_{iqd} \geq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D} \tag{A.34}$$

$$(C_{PUASSiq} V_{ARi} + C_{PUOHiq}) x_{iqdn} - \\ C_{APCPU} o_{dn} \leq 0, \quad \forall d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{A.35}$$

$$\frac{x_{Diqd}}{N_{RMXiq}} - f_{iqd} - \\ \frac{N_{RMXiq}}{N_{Riqr}} (1 - y_{ir}) \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, r \in \mathcal{R}_i \tag{A.36}$$

$$x_{Diqd} \in \mathbb{N}, \quad \forall i \in \mathcal{S}, q \in \{q' \in \mathcal{Q}_i : B_{INDiq'} = 0\}, d \in \mathcal{D} \tag{A.37}$$

$$x_{Diqd} \geq 1, \quad \forall i \in \mathcal{S}, q \in \{q' \in \mathcal{Q}_i : B_{INDiq'} \neq 0\}, \\ d \in \{d' \in \mathcal{D} : B_{INDiq} = d'\} \tag{A.38}$$

$$x_{Diqd} = 0, \quad \forall i \in \mathcal{S}, q \in \{q' \in \mathcal{Q}_i : B_{INDiq'} \neq 0\}, \\ d \in \{d' \in \mathcal{D} : B_{INDiq} \neq d'\} \tag{A.39}$$

$$u_{iqd} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D} \tag{A.40}$$

$$x_{iqdn} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{A.41}$$

$$y_{ir} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, r \in \mathcal{R}_i \tag{A.42}$$

$$o_{dn} \in \{0, 1\}, \quad \forall d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{A.43}$$

A.2.2   Semi-active approach

Objective function

$$
\begin{aligned}
z_{HCESA} = C_{OSTPWR}H_{RS}\Bigg(&\sum_{d\in\mathcal{D}_{PR}}\sum_{n\in\mathcal{N}_d} P_{WRIDLE}o_{dn} \\
&+ P_{WRCOEFF}\sum_{i\in\mathcal{S}}\sum_{q\in\mathcal{Q}_i}\sum_{d\in\mathcal{D}_{PR}}\sum_{n\in\mathcal{N}_d} C_{PUOHiq}x_{iqdn} \\
&+ P_{WRCOEFF}\sum_{i\in\mathcal{S}}\sum_{q\in\mathcal{Q}_i}\sum_{d\in\mathcal{D}_{PR}} C_{PUDEMiq}f_{iqd}\Bigg) \\
&+ H_{RS}\Bigg(\sum_{i\in\mathcal{S}}\sum_{q\in\mathcal{Q}_i}\sum_{d\in\mathcal{D}_{PU}} C_{OSTVMACTiqd}(w_{Diqd}+v_{Diqd}) \\
&+ \sum_{i\in\mathcal{S}}\sum_{q\in\mathcal{Q}_i}\sum_{d\in\mathcal{D}_{PU}} C_{OSTVMPASiqd}(x_{Diqd}-(w_{Diqd}+v_{Diqd}))\Bigg)
\end{aligned}
\tag{A.44}
$$

Constraints

$$
\sum_{r\in\mathcal{R}_i} y_{ir} = 1, \quad \forall i\in\mathcal{S}
\tag{A.45}
$$

$$
\sum_{d\in\mathcal{D}} x_{Diqd} - \sum_{r\in\mathcal{R}_i} N_{Riqr}y_{ir} = 0, \quad \forall i\in\mathcal{S}, q\in\mathcal{Q}_i
\tag{A.46}
$$

$$
\sum_{n\in\mathcal{N}_d} x_{iqdn} - x_{Diqd} = 0, \quad \forall i\in\mathcal{S}, q\in\mathcal{Q}_i, d\in\mathcal{D}_{PR}
\tag{A.47}
$$

$$
\sum_{d\in\mathcal{D}} w_{Diqd} - \sum_{r\in\mathcal{R}_i} A_{CTRiqr}y_{ir} = 0, \quad \forall i\in\mathcal{S}, q\in\mathcal{Q}_i
\tag{A.48}
$$

$$
\sum_{n\in\mathcal{N}_d} w_{iqdn} - w_{Diqd} = 0, \quad \forall i\in\mathcal{S}, q\in\mathcal{Q}_i, d\in\mathcal{D}_{PR}
\tag{A.49}
$$

$$\sum_{d\in\mathcal{D}} v_{Diqd} -$$

$$\sum_{r\in\mathcal{R}_i}\left[S_{ACOEFFiq}\frac{N_{Riqr}-A_{CTRiqr}}{A_{CTRiqr}}\right]y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \qquad (A.50)$$

$$\sum_{n\in\mathcal{N}_d} v_{iqdn} - v_{Diqd} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR} \qquad (A.51)$$

$$w_{Diqd} + v_{Diqd} - x_{Diqd} \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PU} \qquad (A.52)$$

$$w_{iqdn} + v_{iqdn} - x_{iqdn} \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \qquad (A.53)$$

$$\sum_{d\in\mathcal{D}} u_{iqd} - \sum_{r\in\mathcal{R}_i} N_{Riqr}S_{PREADiq}y_{ir} \geq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \qquad (A.54)$$

$$x_{Diqd} - u_{iqd} \geq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D} \qquad (A.55)$$

$$\sum_{i\in\mathcal{S}}\sum_{q\in\mathcal{Q}_i} C_{PUASSiq}V_{ARi}w_{iqdn} +$$

$$\sum_{i\in\mathcal{S}}\sum_{q\in\mathcal{Q}_i} C_{PUASSiq}V_{ARi}v_{iqdn} +$$

$$\sum_{i\in\mathcal{S}}\sum_{q\in\mathcal{Q}_i} C_{PUOHiq}x_{iqdn} - C_{APCPU}o_{dn} \leq 0, \quad \forall d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \qquad (A.56)$$

$$\frac{w_{Diqd}}{A_{CTRiqr}} - f_{iqd} -$$

$$\frac{A_{CTRMXiq}}{A_{CTRiqr}}(1-y_{ir}) \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, r \in \mathcal{R}_i \qquad (A.57)$$

$$x_{Diqd} \in \mathbb{N}, \quad \forall i \in \mathcal{S}, q \in \{q' \in \mathcal{Q}_i : B_{INDiq'} = 0\}, d \in \mathcal{D} \qquad (A.58)$$

$$x_{Diqd} \geq 1, \quad \forall i \in \mathcal{S}, q \in \{q' \in \mathcal{Q}_i : B_{INDiq'} \neq 0\},$$
$$d \in \{d' \in \mathcal{D} : B_{INDiq} = d'\} \qquad (A.59)$$

$$x_{Diqd} = 0, \quad \forall i \in \mathcal{S}, q \in \{q' \in \mathcal{Q}_i : B_{INDiq'} \neq 0\},$$
$$d \in \{d' \in \mathcal{D} : B_{INDiq} \neq d'\} \qquad (A.60)$$

$$w_{Diqd} \in \mathbb{N}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D} \tag{A.61}$$

$$v_{Diqd} \in \mathbb{N}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D} \tag{A.62}$$

$$u_{iqd} \in \{0,1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D} \tag{A.63}$$

$$x_{iqdn} \in \{0,1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{A.64}$$

$$w_{iqdn} \in \{0,1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{A.65}$$

$$v_{iqdn} \in \{0,1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{A.66}$$

$$y_{ir} \in \{0,1\}, \quad \forall i \in \mathcal{S}, r \in \mathcal{R}_i \tag{A.67}$$

$$o_{dn} \in \{0,1\}, \quad \forall d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{A.68}$$

A.2.3   Capacity-lowering approach

Objective function

$$
\begin{aligned}
z_{HCECL} = {} & C_{OSTPWR} H_{RS} \Bigg(\sum_{d \in \mathcal{D}_{PR}} \sum_{n \in \mathcal{N}_d} P_{WRIDLE} o_{dn} \\
& + P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PR}} \sum_{n \in \mathcal{N}_d} C_{PUOHiq} x_{iqdn} \\
& + P_{WRCOEFF} \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PR}} C_{PUDEMiq} f_{iqd} \Bigg) \\
& + H_{RS} \Bigg(\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PU}} C_{OSTVMACTiqd} w_{Diqd} \\
& + \sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} \sum_{d \in \mathcal{D}_{PU}} C_{OSTVMPASiqd} (x_{Diqd} - w_{Diqd}) \Bigg)
\end{aligned}
\tag{A.69}
$$

Constraints

$$\sum_{r \in \mathcal{R}_i} y_{ir} = 1, \quad \forall i \in \mathcal{S} \tag{A.70}$$

$$\sum_{d \in \mathcal{D}} x_{Diqd} - \sum_{r \in \mathcal{R}_i} N_{Riqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{A.71}$$

$$\sum_{n \in \mathcal{N}_d} x_{iqdn} - x_{Diqd} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR} \tag{A.72}$$

$$\sum_{d \in \mathcal{D}} w_{Diqd} - \sum_{r \in \mathcal{R}_i} A_{CTRiqr} y_{ir} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{A.73}$$

$$\sum_{n \in \mathcal{N}_d} w_{iqdn} - w_{Diqd} = 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR} \tag{A.74}$$

$$w_{Diqd} - x_{Diqd} \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PU} \tag{A.75}$$

$$w_{iqdn} - x_{iqdn} \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{A.76}$$

$$\sum_{d \in \mathcal{D}} u_{iqd} - \sum_{r \in \mathcal{R}_i} N_{Riqr} S_{PREADiq} y_{ir} \geq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i \tag{A.77}$$

$$x_{Diqd} - u_{iqd} \geq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D} \tag{A.78}$$

$$m_{dn} -$$
$$C_{PUASSiq} V_{ARi}(x_{iqdn} - w_{iqdn}) \geq 0, \; \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{A.79}$$

$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUASSiq} V_{ARi} w_{iqdn} +$$
$$\sum_{i \in \mathcal{S}} \sum_{q \in \mathcal{Q}_i} C_{PUOHiq} x_{iqdn} - C_{APCPU} o_{dn} \leq 0, \quad \forall d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{A.80}$$

$$\frac{w_{Diqd}}{A_{CTRiqr}} - f_{iqd} -$$
$$\frac{A_{CTRMXiq}}{A_{CTRiqr}}(1 - y_{ir}) \leq 0, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, r \in \mathcal{R}_i \tag{A.81}$$

$$x_{Diqd} \in \mathbb{N}, \quad \forall i \in \mathcal{S}, q \in \{q' \in \mathcal{Q}_i : B_{INDiq'} = 0\}, d \in \mathcal{D} \tag{A.82}$$

$$x_{Diqd} \geq 1, \quad \forall i \in \mathcal{S}, q \in \{q' \in \mathcal{Q}_i : B_{INDiq'} \neq 0\},$$
$$d \in \{d' \in \mathcal{D} : B_{INDiq} = d'\} \tag{A.83}$$

$$x_{Diqd} = 0, \quad \forall i \in \mathcal{S}, q \in \{q' \in \mathcal{Q}_i : B_{INDiq'} \neq 0\},$$
$$d \in \{d' \in \mathcal{D} : B_{INDiq} \neq d'\} \tag{A.84}$$

$$w_{Diqd} \in \mathbb{N}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D} \tag{A.85}$$

$$u_{iqd} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D} \tag{A.86}$$

$$x_{iqdn} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{A.87}$$

$$w_{iqdn} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, q \in \mathcal{Q}_i, d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{A.88}$$

$$y_{ir} \in \{0, 1\}, \quad \forall i \in \mathcal{S}, r \in \mathcal{R}_i \tag{A.89}$$

$$o_{dn} \in \{0, 1\}, \quad \forall d \in \mathcal{D}_{PR}, n \in \mathcal{N}_d \tag{A.90}$$

B   Mosel code

```
!*****************************************************************************************************************!
!*-------------------------------------------------------------------------------------------------------------*!
!*-------------------------------------SERVICE DEPLOYMENT MODEL-------------------------------------------------*!
!*-------------------------------------IN A PRIVATE DATA CENTER-------------------------------------------------*!
!*-------------------------------------------------------------------------------------------------------------*!
!*****************************************************************************************************************!

model sdppdc
uses "mmxprs"; !gain access to the Xpress-Optimizer solver

parameters
    !Data file
    ! Used data files are pdc5.dat, pdc15.dat, pdc20.dat and pdc40.dat
    DATA = 'pdc40.dat'
end-parameters

!-------------------------------------------------*
! DECLARATIONS
!-------------------------------------------------*
declarations
    !-------------------------------------------------*
    !SETS
    !-------------------------------------------------*
    !Set of services, indexed by i
    SERV:        set                            of integer
    !Set of components of service i, indexed by q
    COMP:        array(SERV)                    of range
    !Set of replication patterns for service i, indexed by r
    RSET:        array(SERV) of set             of integer
    !Set of nodes, index by n
    NODES:       set                            of integer

    !-------------------------------------------------*
    !PARAMETERS READ FROM DATA
    !-------------------------------------------------*
    !MODE corresponds to the different approaches
    !MODE = 0 -> Capacity-lowering
    !MODE = 1 -> Semi-active
    !MODE = 2 -> All-active
    MODE:                                       integer
    !SYM corresponds to the strategies used to reduce the number of symmetrical solutions
    !SYM = 0 -> SYM0: only predeployment
    !SYM = 1 -> SYM1: sort nodes according to the amount of CPU power assigned to replicas
    !SYM = 2 -> SYM2: sort nodes according to whether they are turned on or not
    SYM:                                        integer
    !Number of services
    NOS:                                        integer
    !Number of nodes
    NON:                                        integer
    !Number of components for each service
    NOC:         array(SERV)                    of integer
    !Maximum number of replicas of each component
    MAXREP:      dynamic array(SERV, range)     of integer
    !Maximum number of active replicas of each component
    MAXACT:      dynamic array(SERV, range)     of integer
    !CPU capacity for the nodes
    CAPCPU:                                     integer
    !CPU power overhead for all replicas of (i,q)
    CPUOH:       dynamic array(SERV, range)     of real
    !Cost of energy usage per hour
    CPWR:                                       real
    !Length of the time period
    HRS:                                        real
    !Converting CPU power to power usage
    PWRCOEFF:                                   real
    !Minimum power needed for operation of a node
    PWRIDLE:                                    real
    !Stationary availability of each component with no replication
    AVAILCOMP:   dynamic array(SERV, range)     of real
    !Required availability for each service
    REQAVAIL:    array(SERV)                    of real
    !Required response time for each service
    REQRESPT:    array(SERV)                    of integer
    !The avg. CPU power needed to handle a request for service i
    JOBLOAD:     dynamic array(SERV, range)     of real
    !The avg. number of request for service i per time unit
    DEM:         array(SERV)                    of integer
    !The parameter ensuring that a service the peaks in demand
    VAR:         array(SERV)                    of real
    !The amount of CPU power assigned to an active (or semiactive) replica
    CPUASS:      dynamic array(SERV, range)     of real
    !Parameter for controlling the number of semi-active replicas
    SACOEFF:     dynamic array(SERV, range)     of real


    !-------------------------------------------------*
```

125

```
    ! PARAMETERS TO CALCULATE
    !----------------------------------------------*
    !The number of replication settings for each service
    RMAX:         array(SERV)                     of integer
    !Stationary availability of each component using replication pattern r
    AVCOMPREAL: dynamic array(SERV, range, range)  of real
    !Stationary availability of a service given a replication pattern r
    AVAIL:        array(SERV, range)               of real
    !Number of replicas of a comp given a replication setting
    NR:           dynamic array(SERV, range, range)  of integer
    !Number of active replicas of a comp given a rep setting
    ACTR:         dynamic array(SERV, range, range)  of integer
    !The minimum number of active replicas for a (i,q)
    ACTRMN:       dynamic array(SERV, range)        of integer
    !Response time for each component of each service
    RESPT:        dynamic array(SERV, range)        of integer
    RTTOT:        dynamic array(SERV, range)        of real
    !The demanded CPU power from requests
    CPUDEM:       dynamic array(SERV, range)        of real
    !Data structure used to predeploy the components which has the highest value of ACTRMN
    PREDEP:       record
                     i,q,a:       integer
                  end-record

    !----------------------------------------------*
    !VARIABLES
    !----------------------------------------------*
    != 1 if a replica of (i,q) is deployed on node n, = 0 otherwise, cf. x_iqn
    mapping:   dynamic array(SERV, range, NODES)   of mpvar
    != 1 if a replica of (i,q) is active on node n, = 0 otherwise, cf. w_iqn
    active:    dynamic array(SERV, range, NODES)   of mpvar
    != 1 if a replica of (i,q) is semi-active on node n, = 0 otherwise, cf. v_iqn
    semiact:   dynamic array(SERV, range, NODES)   of mpvar
    != 1 if replication pattern r is used for service i, = 0 otherwise, cf. y_ir
    repset:    dynamic array(SERV, range)          of mpvar
    != 1 if node n is turned on, = 0 otherwise, cf. o_n
    used:         array(NODES)                     of mpvar
    !The amount of non-assignable amount of CPU power on node n, cf. m_n
    maxrepcap: array(NODES)                        of mpvar
    !The amount of CPU power assigned to the replicas of a node, cf. s_n (used for sorting in SYM1)
    assignedc: array(NODES)                        of mpvar

    !----------------------------------------------*
    !CONSTRAINTS
    !----------------------------------------------*
    !Ensure that one and only one replication pattern is used for each service
    Repsetsum:    array(SERV)                     of linctr
    !Ensure that all replicas are deployed
    Allreps:   dynamic array(SERV, range)         of linctr
    !Ensure that there are enough active replicas
    Allactive: dynamic array(SERV, range)         of linctr
    !Ensure that there are enough semi-active replicas
    Allsemiact: dynamic array(SERV, range)        of linctr
    !Ensure that a if a replica is active (or semi-active) it has to be deployed on that node
    Activeif:  dynamic array(SERV, range, NODES)  of linctr
    !Set the non-assignable amount of CPU power on each node
    Setmxrepcap:dynamic array(SERV, range, NODES)  of linctr
    !CPU capacity constraint on all nodes
    Cpucap:       array(NODES)                     of linctr
    !Set the amount of CPU power assigned to the replicas on a node
    Setass:       array(NODES)                     of linctr
    !Sort the nodes according to assignedc
    Sortass:      array(NODES)                     of linctr
    !Sort the nodes according to used
    Sortused:     array(NODES)                     of linctr
    !----------------------------------------------*
    !OBJECTIVE
    !----------------------------------------------*
    !Objective function: cost of energy usage
    Objective:                    linctr

end-declarations

!----------------------------------------------*
! Initializations
!----------------------------------------------*
writeln("Datafile = ",DATA)
initializations from DATA
    NOS NON CPWR HRS MODE SYM
end-initializations

!Initialize set of services and set of nodes
SERV     := 1..NOS
NODES    := 1..NON

initializations from DATA
    NOC CAPCPU REQAVAIL REQRESPT PWRCOEFF PWRIDLE DEM VAR
end-initializations
```

126

```
!Initialize the set of components of each service
forall(i in SERV) COMP(i) := 1..NOC(i)

initializations from DATA
    [MAXREP, MAXACT, AVAILCOMP, SACOEFF] AS "REPDATA"
    [JOBLOAD, CPUASS, CPUOH] AS "LOADDATA"
end-initializations

!Calculate CPUDEM(i,q)
forall(i in SERV, q in COMP(i)) do
    CPUDEM(i,q) := DEM(i)*JOBLOAD(i,q)
end-do


!--------------------------------------------------*
! PROCEDURES
!--------------------------------------------------*
forward function doFactorial(x:integer):integer
forward procedure createReplicationPatterns
forward function calcAvail(N:array(SERV, range) of integer, A:array(SERV, range) of integer, i:integer):real
forward function calcRespt(A:array(SERV, range) of integer, i:integer):real
forward function checkStp(TMP:array(SERV, range) of integer, UP:array(SERV,range) of integer, i:integer):boolean
forward function checkAllGeqA(TMPA:array(SERV, range) of integer, i:integer):boolean
forward function checkAllGeqN(TMPN:array(SERV, range) of integer, TMPA:array(SERV, range) of integer, i:integer):boolean

!--------------------------------------------------*
! PROCEDURE TO GENERATE REPLICATION PATTERNS
!--------------------------------------------------*
procedure createReplicationPatterns
    declarations
        !Temporary total number of replicas
        TMPN:    array(SERV, range)  of integer
        !Temporary number of active replicas
        TMPA:    array(SERV, range)  of integer
        !Temporary upper bound on the number of active replicas
        AUP:     array(SERV, range)  of integer
        !Temporary upper bound on the total number of replicas
        NUP:     array(SERV, range)  of integer
        !Temporary lower bound on the number of active replicas
        ALW:     array(SERV, range)  of integer
        !Temporary lower bound on the total number of replicas
        NLW:     array(SERV, range)  of integer
        !Current replication pattern
        r:                           integer
    end-declarations
    !Do forall services
    forall(i in SERV) do
        !Initialize r = 1
        r := 1
        !Initialize the number of active replicas
        forall(q in COMP(i)) do
            TMPA(i,q) := 1
            AUP(i,q) := MAXACT(i,q)
            ALW(i,q) := 1
        end-do
        !FRISTCHCKA used to stop the loop if one needs only 1 active replica of each component
        FIRSTCHCKA := true
        !Outer loop
        while(checkStp(TMPA,AUP,i)) do
            !Check response time requirement and if there exists a clearly better AC already
            if(calcRespt(TMPA,i) <= REQRESPT(i) and checkAllGeqA(TMPA,i)) then
                !Initialize the total number of replicas
                forall(q in COMP(i)) do
                    TMPN(i,q) := TMPA(i,q)
                    NUP(i,q) := MAXREP(i,q)
                    NLW(i,q) := TMPA(i,q)
                end-do
                !Lower temporary upper bound if only one component have another number of active replicas than 1
                AUPCHCK := 0
                forall(q in COMP(i) | TMPA(i,q) <> 1) AUPCHCK += 1
                if(AUPCHCK = 1) then
                    forall(q in COMP(i) | TMPA(i,q) <> 1) AUP(i,q) := TMPA(i,q)
                end-if
                !FRISTCHCKN used to stop the loop if one needs no passive replicas of a component
                FIRSTCHCKN := true
                !Inner loop
                while(checkStp(TMPN,NUP,i)) do
                    !Check availability requirement and if there exists a clearly better TC, based on the same AC
                    if(calcAvail(TMPN,TMPA,i) >= REQAVAIL(i) and checkAllGeqN(TMPN, TMPA, i)) then
                        !Update NR, ACTR, the availabilies, the response time and RSET
                        forall(q in COMP(i)) do
                            NR(i,q,r) := TMPN(i,q)
                            ACTR(i,q,r) := TMPA(i,q)
                        end-do
                        AVAIL(i,r) := calcAvail(TMPN,TMPA,i)
                        RTTOT(i,r) := calcRespt(TMPA,i)
                        RSET(i) += {r}
```

```
                            RMAX(i) := r
                            !Increment r
                            r += 1
                            !Break loop if FRISTCHCKN is true
                            if(FIRSTCHCKN) then
                                break
                            else
                                !Lower temporary upper bound if only one component have another # passive replicas than 1
                                NUPCHCK := 0
                                forall(q in COMP(i) | TMPN(i,q) <> TMPA(i,q)) NUPCHCK += 1
                                if(NUPCHCK = 1) then
                                    forall(q in COMP(i) | TMPN(i,q) <> TMPA(i,q)) NUP(i,q) := TMPN(i,q)
                                end-if
                            end-if
                    end-if !REQAVAIL
                    FIRSTCHCKN := false
                    !Find new TMPN(i)
                    q := NOC(i)
                    while(q > 1) do
                        qq := NOC(i)
                        while(qq > 0) do
                            if(TMPN(i,qq) < NUP(i,qq)) then
                                TMPN(i,qq) += 1
                                break 2
                            else
                                TMPN(i,qq) := NLW(i,qq)
                                if(qq = 1) then
                                    NLW(i,q) += 1
                                    TMPN(i,q) := NLW(i,q)
                                    if(q = 1) then
                                        break 2
                                    else
                                        qq := NOC(i) - q
                                        q -= 1
                                    end-if
                                else
                                    qq -= 1
                                end-if
                            end-if
                            !qq -= 1
                        end-do  !qq
                    end-do !q
                end-do !checkStp(TMPN,NUP,i)
                if(FIRSTCHCKA) then
                    break
                end-if
            end-if !REQRESPT
            FIRSTCHCKA := false
            !Find new TMPA(i)
            q := NOC(i)
            while(q > 1) do
                qq := NOC(i)
                while(qq > 0) do
                    if(TMPA(i,qq) < AUP(i,qq)) then
                        TMPA(i,qq) += 1
                        break 2
                    else
                        TMPA(i,qq) := ALW(i,qq)
                        if(qq = 1) then
                            !NLW(i,qq) += 1
                            ALW(i,q) += 1
                            TMPA(i,q) := ALW(i,q)

                            if(q = 1) then
                                break 2
                            else
                                qq := NOC(i) - q
                                q -= 1
                            end-if
                        else
                            qq -= 1
                        end-if
                    end-if
                end-do  !qq
            end-do !q
        end-do !checkStp(TMPA,AUP,i)
    end-do !forall i
end-procedure

!Calculates the response time based on TMPA for i
function calcRespt(A:array(SERV, range) of integer, i:integer):real
    !Check if stable queue
    INF := false
    forall(q in COMP(i)) do
        if(CPUASS(i,q)/JOBLOAD(i,q) - DEM(i)/A(i,q) <= 0) then
            INF := true
        end-if
    end-do
```

```
    if(INF = false) then
        !Multiply be 1000 to get milliseconds
        returned := sum(q in COMP(i)) 1000/(CPUASS(i,q)/JOBLOAD(i,q) - DEM(i)/A(i,q))
    else
        !Else return a very high number
        returned := 1000000000
    end-if
end-function


!Calculate the service availability based on TMPA and TMPN for i
function calcAvail(N:array(SERV, range) of integer, A:array(SERV, range) of integer, i:integer):real
    returned := prod(q in COMP(i)) sum(k in A(i,q)..N(i,q))(doFactorial(N(i,q))/(doFactorial(k)*doFactorial(N(i,q)-k)))*
                                            AVAILCOMP(i,q)^k*(1-AVAILCOMP(i,q))^(N(i,q)-k)
end-function


!Chech if loop should stop
function checkStp(TMP:array(SERV, range) of integer, UP:array(SERV,range) of integer, i:integer):boolean
    retval := false
    forall(q in COMP(i) | TMP(i,q) <> UP(i,q)) retval := true
    returned := retval
end-function


!Check if there exists a clearly better AC already
function checkAllGeqA(TMPA:array(SERV, range) of integer, i:integer):boolean
    retval := true
    forall(r in RSET(i)) do
        check := 0
        forall(q in COMP(i)) do
            if(TMPA(i,q) >= ACTR(i,q,r)) then
                check += 1
            end-if
        end-do
        if(check = NOC(i)) then
            retval := false
            break
        end-if
    end-do
    returned := retval
end-function


!Check if there exists a clearly better TC, based on the same AC
function checkAllGeqN(TMPN:array(SERV, range) of integer, TMPA:array(SERV, range) of integer, i:integer):boolean
    declarations
        !Set of replication patterns that is based on the same AC, cf. R*
        TMPRSET:set of integer
    end-declarations
    retval := true
    !Calculate R*
    forall(r in RSET(i)) do
        checkr := 0
        forall(q in COMP(i)) do
            if(TMPA(i,q) = ACTR(i,q,r)) then
                checkr += 1
            end-if
        end-do
        if(checkr = NOC(i)) then
            TMPRSET += {r}
        end-if
    end-do
    forall(r in TMPRSET) do
        checkn := 0
        forall(q in COMP(i)) do
            if(TMPN(i,q) >= NR(i,q,r)) then
                checkn += 1
            end-if
        end-do
        if(checkn = NOC(i)) then
            retval := false
            break
        end-if
    end-do
    returned := retval
end-function


!doFacorial - Function which computes the factorial of an integer
function doFactorial(x:integer):integer
        if (x <= 1) then
            returned:=1;
        else
            returned:=(x * doFactorial(x-1));
        end-if
end-function


writeln("createReplicationPatterns BEGIN")
createReplicationPatterns
writeln("createReplicationPatterns DONE")
```

```
!Calculate ACTRMN(i,q)
forall(i in SERV, q in COMP(i)) do
    ACTRMN(i,q) := MAXACT(i,q)
    forall(r in RSET(i)) do
        if(ACTR(i,q,r) < ACTRMN(i,q)) then
            ACTRMN(i,q) := ACTR(i,q,r)
        end-if
    end-do
end-do

!Print out replication settings
forall(i in SERV) do
    writeln("--------SERVICE ",i," -------------")
    write("r ")
    forall(q in COMP(i))do
        write("a_",q," n_",q,"    ")
    end-do
    writeln("AVAIL(",i,",r)                    RTTOT(",i,",r)")
    forall(r in RSET(i)) do
        write(r," ")
        forall(q in COMP(i)) do
            write(" ",ACTR(i,q,r),"    ",NR(i,q,r),"      ")
        end-do
        writeln(AVAIL(i,r)," >= ",REQAVAIL(i),"        ",RTTOT(i,r)," <= ",REQRESPT(i))
    end-do
end-do

!-------------------------------------------------*
!CREATE VARIABLES
!-------------------------------------------------*


forall(i in SERV, q in COMP(i), n in NODES) do
    !mapping(i,q,n)
    create(mapping(i,q,n))
    mapping(i,q,n) is_binary
    if(MODE <> 2) then
        !active(i,q,n)
        create(active(i,q,n))
        active(i,q,n) is_binary
        if(MODE = 1) then
            !semiact(i,q,n)
            create(semiact(i,q,n))
            semiact(i,q,n) is_binary
        end-if
    end-if
end-do

!repset(i,r)
forall(i in SERV, r in RSET(i)) do
    create(repset(i,r))
    repset(i,r) is_binary
end-do

forall(n in NODES) do
    !used(n)
    create(used(n))
    used(n) is_binary
    if(MODE = 0) then
        !maxrepcap(n)
        create(maxrepcap(n))
    end-if
    if(SYM = 1) then
        !assignedc(n)
        create(assignedc(n))
    end-if
end-do


!-------------------------------------------------*
! DEFINE OBJECTIVE FUNCTION
!-------------------------------------------------*
Objective :=        CPWR * HRS*(sum(i in SERV, q in COMP(i))PWRCOEFF*CPUDEM(i,q) +
                    sum(i in SERV, q in COMP(i), n in NODES)PWRCOEFF*CPUOH(i,q) * mapping(i,q,n) +
                                            sum(n in NODES)PWRIDLE * used(n))

!-------------------------------------------------*
! CONSTRAINTS
!-------------------------------------------------*

if(MODE = 0) then
    !*****************************************************************************************
    !BEGIN CAPACITY LOWERING APPROACH
    !*****************************************************************************************
    !Choose one and only one replication pattern for each service
    forall(i in SERV) Repsetsum(i) := sum(r in RSET(i)) repset(i,r) = 1
```

130

```
!Distribute the replica to the nodes
forall(i in SERV, q in COMP(i)) Allreps(i,q) :=                    sum(n in NODES)   mapping(i,q,n) -
                                          sum(r in RSET(i)) NR(i,q,r) * repset(i,r) = 0


!Set ACTR replicas to be active
forall(i in SERV, q in COMP(i)) Allact(i,q) :=                     sum(n in NODES)   active(i,q,n) -
                                          sum(r in RSET(i)) ACTR(i,q,r) * repset(i,r) = 0


!Ensure that a replica of (i,q) only can be active on a node if there is a replica of (i,q) deployed there
forall(i in SERV, q in COMP(i), n in NODES) Activeif(i,q,n) := active(i,q,n) -
                                               mapping(i,q,n) <= 0


!Set non-assignable CPU power on the nodes
forall(n in NODES, i in SERV, q in COMP(i)) do
    Setmxrepcap(i,q,n) :=                      maxrepcap(n) -
                    CPUASS(i,q)*VAR(i) * (mapping(i,q,n) - active(i,q,n)) >= 0

end-do

if(SYM = 1) then
    !Calculate assingedc(n)
    forall(n in NODES) do
        Setass(n) := sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i) * active(i,q,n) +
                              sum(i in SERV, q in COMP(i))CPUOH(i,q) * mapping(i,q,n) -
                                                    assignedc(n) = 0
    end-do

    !Capacity constraints on the nodes
    forall(n in NODES) do
        Cpucap(n) :=          assignedc(n) +
                              maxrepcap(n) -
                  CAPCPU * used(n) <= 0
    end-do
else
    !Capacity constraints on the nodes
    forall(n in NODES) do
        Cpucap(n) := sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i)) * active(i,q,n) +
                              sum(i in SERV, q in COMP(i))CPUOH(i,q) * mapping(i,q,n) +
                                                    maxrepcap(n) -
                                          CAPCPU * used(n) <= 0
    end-do
end-if
!********************************************************************************************
!END CAPACITY LOWERING APPROACH
!********************************************************************************************

elif(MODE = 1) then
    !********************************************************************************************
    !BEGIN SEMI-ACTIVE APPROACH
    !********************************************************************************************

    !Choose one and only one replication pattern for each service
    forall(i in SERV) Repsetsum(i) := sum(r in RSET(i)) repset(i,r) = 1


    !Distribute the replica to the nodes
    forall(i in SERV, q in COMP(i)) Allreps(i,q) :=                    sum(n in NODES)   mapping(i,q,n) -
                                              sum(r in RSET(i)) NR(i,q,r) * repset(i,r) = 0


    !Set ACTR replicas to be active
    forall(i in SERV, q in COMP(i)) Allact(i,q) :=                     sum(n in NODES)   active(i,q,n) -
                                              sum(r in RSET(i)) ACTR(i,q,r) * repset(i,r) = 0

    !Set the right number of replicas to be semi-active
    forall(i in SERV, q in COMP(i)) do
        Allsemiact(i,q) :=                                      sum(n in NODES)   semiact(i,q,n)-
                        sum(r in RSET(i))ceil(SACOEFF(i,q)*(NR(i,q,r)-ACTR(i,q,r))/ACTR(i,q,r)) * repset(i,r) = 0
    end-do

    !Ensure that a replica of (i,q) only can be active or semi-active on a node
    !if there is a replica of (i,q) deployed there
    forall(i in SERV, q in COMP(i), n in NODES) Activeif(i,q,n) :=  active(i,q,n) +
                                                 semiact(i,q,n) -
                                                 mapping(i,q,n) <= 0


    if(SYM = 1) then
        !Calculate assingedc(n)
        forall(n in NODES) do
            Setass(n) := sum(i in SERV, q in COMP(i))CPUASS(i,q)*VAR(i) * active(i,q,n) +
                        sum(i in SERV, q in COMP(i))CPUASS(i,q)*VAR(i) * semiact(i,q,n) +
                            sum(i in SERV, q in COMP(i))CPUOH(i,q) * mapping(i,q,n) -
                                                    assignedc(n) = 0
        end-do
        !Capacity constraints on the nodes
        forall(n in NODES) do
            Cpucap(n) :=          assignedc(n) -
                        CAPCPU * used(n) <= 0
        end-do
```

```
    else
        !Capacity constraints on the nodes
        forall(n in NODES) do
            Cpucap(n) := sum(i in SERV, q in COMP(i))CPUASS(i,q)*VAR(i) * active(i,q,n) +
                         sum(i in SERV, q in COMP(i))CPUASS(i,q)*VAR(i) * semiact(i,q,n) +
                              sum(i in SERV, q in COMP(i))CPUOH(i,q) * mapping(i,q,n) -
                                                       CAPCPU * used(n) <= 0
        end-do
    end-if

    !*********************************************************************************************
    !END SEMI-ACTIVE APPROACH
    !*********************************************************************************************
elif(MODE = 2) then
    !*********************************************************************************************
    !BEGIN ALL-ACTIVE APPROACH
    !*********************************************************************************************

    !Choose one and only one replication pattern for each service
    forall(i in SERV) Repsetsum(i) := sum(r in RSET(i)) repset(i,r) = 1

    !Distribute the replica to the nodes
    forall(i in SERV, q in COMP(i)) Allreps(i,q) :=                sum(n in NODES)   mapping(i,q,n) -
                                           sum(r in RSET(i)) NR(i,q,r) * repset(i,r) = 0


    if(SYM = 1) then
        !Set assignedc(n)
        forall(n in NODES) do
            Setass(n) := sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i) + CPUOH(i,q)) * mapping(i,q,n) -
                                                                    assignedc(n) = 0
        end-do

        !Capacity constraints on the nodes
        forall(n in NODES) do
            Cpucap(n) :=            assignedc(n) -
                         CAPCPU * used(n) <= 0
        end-do
    else
!Capacity constraints on the nodes
        forall(n in NODES) do
            Cpucap(n) := sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i) + CPUOH(i,q)) * mapping(i,q,n) -
                                                           CAPCPU * used(n) <= 0
        end-do
    end-if

    !*********************************************************************************************
    !END ALL-ACTIVE APPROACH
    !*********************************************************************************************
end-if


!---------------------------------------------------*
!Doing some predefined decisions in order to lower the number of symmetrical solutions
!---------------------------------------------------*

!Sort nodes in order to avoid symmetrical solutions
if(SYM = 1) then
    forall(n in NODES | n < NON) Sortass(n) := assignedc(n) - assignedc(n+1) >= 0
elif(SYM = 2) then
    forall(n in NODES | n < NON) Sortused(n) := used(n) - used(n+1) >= 0
end-if

if(SYM <> 1) then
    !The component with the highest ACTRMN is deployed on the nodes with the lowest index
    PREDEP.i := 1
    PREDEP.q := 1
    PREDEP.a := ACTRMN(1,1)
    forall(i in SERV, q in COMP(i) | PREDEP.a < ACTRMN(i,q)) do
        PREDEP.i := i
        PREDEP.q := q
        PREDEP.a := ACTRMN(i,q)
    end-do
    writeln("(",PREDEP.i,",",PREDEP.q,") is active on nodes 1..",PREDEP.a)
    forall(n in NODES | n <= PREDEP.a) do
        if(MODE <> 2) then
            active(PREDEP.i, PREDEP.q, n) = 1
        else
            mapping(PREDEP.i, PREDEP.q, n) = 1
        end-if
    end-do
end-if


writeln("")
writeln("Begin running model")
```

132

```
!-------------------------------------------------*
!MINIMIZE OBJECTIVE
!-------------------------------------------------*
!setparam('XPRS_TREEMEMORYLIMIT',1000)
setparam('xprs_verbose',true)
setparam('xprs_miplog', -1000)
setparam('xprs_maxtime',-4000)

minimize(Objective)
writeln("Objective = ",getobjval)
writeln(CPWR*HRS*(sum(i in SERV, q in COMP(i))PWRCOEFF*CPUDEM(i,q)))
writeln(CPWR*HRS*sum(i in SERV, q in COMP(i), n in NODES)PWRCOEFF*CPUOH(i,q) * mapping(i,q,n).sol)
writeln(CPWR*HRS*sum(n in NODES)PWRIDLE * used(n).sol)

!Print out the number of replciations settings for each service
forall(i in SERV) do
    if(RMAX(i) > 0) then
        writeln("Service ",i," has ",RMAX(i)," replication settings that confins with the SLA requirements")
    else
        writeln("Service ",i," has none replication settings: THE MODEL IS INFEASIBLE!")
    end-if
end-do

!Print out information about availability
writeln("")
writeln("Availability print out")
forall(i in SERV, r in RSET(i) | repset(i,r).sol > 0.1) do
    writeln("-------- SERVICE ",i,"----------")
    writeln("Replication setting ",r," is used")
    writeln("AVAIL(",i,",",r,") = ",AVAIL(i,r)," >= ",REQAVAIL(i)," = REQAVAIL(",i,")")
    forall(q in COMP(i)) do
        writeln("NR(",i,",",q,",",r,") = ",NR(i,q,r))
        writeln("ACTR(",i,",",q,",",r,") = ",ACTR(i,q,r))
    end-do
end-do

!Print out Demand and CPU data
writeln("")
writeln("Demand and CPU data print out")
forall(i in SERV) do
    writeln("-------- SERVICE ",i,"----------")
    writeln("Demand for service ",i,": ",DEM(i)," req/sec")
    writeln("VAR(",i,") = ",VAR(i))
    forall(q in COMP(i)) do
        writeln("*Component ",q,"*")
        writeln("JOBLOAD(",i,",",q,") = ",JOBLOAD(i,q))
        writeln("CPUDEM(",i,",",q,") = ",CPUDEM(i,q))
        writeln("CPUASS(",i,",",q,") = ",CPUASS(i,q))
        writeln("Assigned CPU power including VAR(",i,") to each active replica: ",CPUASS(i,q)*VAR(i))
        writeln("CPUASS(",i,",",q,") * Active reps = ",CPUASS(i,q)," * ",sum(r in RSET(i))ACTR(i,q,r)*repset(i,r).sol,
            " = ",CPUASS(i,q) * sum(r in RSET(i))ACTR(i,q,r)*repset(i,r).sol)
        writeln("This gives a component response time of ",
            1000/(CPUASS(i,q)/JOBLOAD(i,q) - DEM(i)/(sum(r in RSET(i))ACTR(i,q,r)*repset(i,r).sol))," ms")
    end-do
end-do

!Print out information about deployment decisions
writeln("")
writeln("Deployment variables, mapping(i,q,n) and active(i,q,n) print out")
forall(i in SERV, q in COMP(i), n in NODES | mapping(i,q,n).sol > 0 or active(i,q,n).sol > 0) do
    write("mapping(",i,",",q,",",n,") = ",mapping(i,q,n).sol," ")
    if(active(i,q,n).sol > 0.5) then
        write("(active)")
    end-if
    if(semiact(i,q,n).sol > 0.5 and MODE = 1) then
        write("(semiactive)")
    end-if
    writeln("")
end-do

!Print out information about CPU capacity and power usage on the nodes
if(MODE <> 2) then
    writeln("")
    writeln("Node capacity print out")
    forall(n in NODES) do
        write("--------- NODE ",n," ")
        if (used(n).sol > 0.5) then
            write("(ON) ")
        else
            write("(OFF) ")
        end-if
        writeln("---------")
        forall(i in SERV, q in COMP(i) | mapping(i,q,n).sol>0.5) do
            write("(",i,",",q,") with CPU load ",(CPUASS(i,q)*VAR(i))*active(i,q,n).sol +
                (CPUASS(i,q)*VAR(i))*semiact(i,q,n).sol + CPUOH(i,q)*mapping(i,q,n).sol," ")
            if(active(i,q,n).sol > 0.5) then
                write("(active)")
            end-if
```

```
            if(semiact(i,q,n).sol > 0.5) then
                write("(semiactive)")
            end-if
            writeln("")
        end-do
        write("Total load on node ",n," = ",sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i))*active(i,q,n).sol+
            sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i))*semiact(i,q,n).sol+
            sum(i in SERV, q in COMP(i))CPUOH(i,q)*mapping(i,q,n).sol)
            if(MODE = 0) then
                writeln(" <= CAPCPU - maxrepcap(",n,") = ",CAPCPU," - ",maxrepcap(n).sol," = ",CAPCPU - maxrepcap(n).sol)
            elif(MODE = 1) then
                writeln(" <= CAPCPU = ",CAPCPU)
            end-if
        writeln("assignedc(",n,") = ",assignedc(n).sol)
    end-do
else
    writeln("")
    writeln("Node capacity print out")
    forall(n in NODES) do
        write("--------- NODE ",n," ")
        if (used(n).sol > 0.5) then
            write("(ON) ")
        else
            write("(OFF) ")
        end-if
        writeln("---------")
        forall(i in SERV, q in COMP(i) | mapping(i,q,n).sol>0.5) do
            writeln("(",i,",",q,") with CPU load ",(CPUASS(i,q)*VAR(i)+CPUOH(i,q))*mapping(i,q,n).sol)
        end-do
        writeln("Total load on node ",n," = ",sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i)+
            CPUOH(i,q))*mapping(i,q,n).sol," <= CAPCPU = ",CAPCPU)
        writeln("assignedc(",n,") = ",assignedc(n).sol)
    end-do
end-if
!Print out information about utilization
if(MODE = 0 or MODE = 1) then
    writeln("")
    forall(n in NODES) do
        writeln("")
        forall(i in SERV, q in COMP(i) | mapping(i,q,n).sol > 0.1) do
            writeln("(",i,",",q,")'s contribution to the utilization on node ",n," is: ",CPUOH(i,q)*mapping(i,q,n).sol +
                CPUDEM(i,q)*sum(r in RSET(i))active(i,q,n).sol/ACTR(i,q,r)*repset(i,r).sol)
        end-do
        writeln("The total utilization on node ",n," is: ",
            sum(i in SERV, q in COMP(i))CPUDEM(i,q)*sum(r in RSET(i))active(i,q,n).sol/ACTR(i,q,r)*repset(i,r).sol +
            sum(i in SERV, q in COMP(i))CPUOH(i,q)*mapping(i,q,n).sol)
    end-do
    writeln("The average utilization on turned on nodes = ",(sum(i in SERV, q in COMP(i), n in NODES)CPUDEM(i,q)*
            sum(r in RSET(i))active(i,q,n).sol/ACTR(i,q,r)*repset(i,r).sol +
            sum(i in SERV, q in COMP(i), n in NODES)CPUOH(i,q)*mapping(i,q,n).sol)/(sum(n in NODES)used(n).sol))
elif(MODE = 2) then
    writeln("")
    forall(n in NODES) do
        writeln("")
        forall(i in SERV, q in COMP(i) | mapping(i,q,n).sol > 0.1) do
            writeln("(",i,",",q,")'s contribution to the utilization on node ",n," is: ",
                (CPUOH(i,q))*mapping(i,q,n).sol +
                CPUDEM(i,q)*sum(r in RSET(i))mapping(i,q,n).sol/NR(i,q,r)*repset(i,r).sol)
        end-do
        writeln("The total utilization on node ",n," is: ",
            sum(i in SERV, q in COMP(i))CPUDEM(i,q)*sum(r in RSET(i))mapping(i,q,n).sol/NR(i,q,r)*repset(i,r).sol +
            sum(i in SERV, q in COMP(i))(CPUOH(i,q))*mapping(i,q,n).sol)
    end-do
    writeln("The average utilization on turned on nodes = ",(sum(i in SERV, q in COMP(i), n in NODES)CPUDEM(i,q)*
            sum(r in RSET(i))mapping(i,q,n).sol/NR(i,q,r)*repset(i,r).sol +
            sum(i in SERV, q in COMP(i), n in NODES)(CPUOH(i,q))*mapping(i,q,n).sol)/
            (sum(n in NODES)used(n).sol))
end-if
!exportprob(EP_MIN, '../lp-files/minpower_fixedrespt.lp',Minpower)

writeln("End running model")

end-model
```

134

```
!*****************************************************************************************************************!
!*-------------------------------------------------------------------------------------------------------------*!
!*----------------------------------SERVICE DEPLOYMENT MODEL---------------------------------------------------*!
!*-------------------------------IN A HYBRID CLOUD ENVIRONMENT-------------------------------------------------*!
!*-------------------------------------------------------------------------------------------------------------*!
!*****************************************************************************************************************!

model sdphce
uses "mmxprs"; !gain access to the Xpress-Optimizer solver

parameters
    !Data file
    ! Used data files are hce5.dat, hce10.dat and hce20.dat
    DATA = 'hce20.dat'
end-parameters

!--------------------------------------------------*
! DECLARATIONS
!--------------------------------------------------*
Declarations

    !--------------------------------------------------*
    !SETS
    !--------------------------------------------------*
    !Set of services, indexed by i
    SERV:        set                                of integer
    !Set of components of service i, indexed by q
    COMP:        array(SERV)                        of range
    !Set of replication patterns for service i, indexed by r
    RSET:        array(SERV) of set                 of integer
    !Set of cloud domains for service deployment, indexed by d
    DOM:         set                                of integer
    !Set of private cloud domains, indexed by d
    DOMPR:       set                                of integer
    !Set of public cloud domain, indexed by d
    DOMPU:       set                                of integer
    !Set of VM types in public cloud domains, indexed by v
    VMS:         array(DOMPU) of set                of integer
    !Set of nodes in private cloud domains, indexed by n
    NODES:       array(DOMPR) of set                of integer

    !--------------------------------------------------*
    !PARAMETERS READ FROM DATA
    !--------------------------------------------------*

    !MODE corresponds to the different approaches
    !MODE = 0 -> Capacity-lowering
    !MODE = 1 -> Semi-active
    !MODE = 2 -> All-active
    MODE:                                           integer
    !SYM corresponds to the strategies used to reduce the number of symmetrical solutions
    !SYM = 0 -> SYM0: only predeployment
    !SYM = 1 -> SYM1: sort nodes according to the amount of CPU power assigned to replicas
    !SYM = 2 -> SYM2: sort nodes according to whether they are turned on or not
    SYM:                                            integer
    !Number of services
    NOS:                                            integer
    !Number of private cloud domains
    NOCLPR:                                         integer
    !Number of public cloud domains
    NOCLPU:                                         integer
    !Number of components for each service
    NOC:         array(SERV)                        of integer
    !Number of different VM types in a public cloud domain
    NOVM:        array(DOMPU)                       of integer
    !Number of nodes in each private cloud domain
    NON:         array(DOMPR)                       of integer
    !Maximum number of replicas of each component
    MAXREP:      dynamic array(SERV, range)         of integer
    !Maximum number of active replicas of each component
    MAXACT:      dynamic array(SERV, range)         of integer
    !Stationary availability of each component with no replication
    AVAILCOMP:   dynamic array(SERV, range)         of real
    !Required availability for each service
    REQAVAIL:    array(SERV)                         of real
    !Required response time for each service
    REQRESPT:    array(SERV)                         of integer
    !Parameter for controlling the number of semi-active replicas
    SACOEFF:     dynamic array(SERV, range)         of real
    !The required number of domains used by a replica in percentage of the total nnumber of replicas
    SPREAD:      dynamic array(SERV, range)         of real
    !=d, if (i,q) is bound to be deployed in cloud domain d, =0 if no restriction
    BINDING:     dynamic array(SERV, range)         of integer
    !Length of time period
    HRS:                                            real
    !Cost of energy usage per time unit
    CPWR:                                            real
```

135

```
!Converting CPU power to power usage
PWRCOEFF:                                          real
!Minimum power needed for operation of a node
PWRIDLE:                                           real
!CPU capacity for the nodes in the private cloud domains
CAPCPU:                                            integer
!CPU-power overhead for replicas
CPUOH:       dynamic array(SERV, range)            of real
!The fixed amount of CPU power assigned to an active (or semiactive) replica
CPUASS:      dynamic array(SERV, range)            of real
!The avg. CPU power needed to handle a request for service i
JOBLOAD:     dynamic array(SERV, range)            of real
!The avg. number of request for service i per time unit
DEM:         array(SERV)                           of real
!The parameter ensuring that a service handles the peaks in demand
VAR:         array(SERV)                           of real
!Cost of deploying a replica in a public cloud domain per hour, using a given VM
COSTVM:      dynamic array(DOMPU, range)           of real
!The size of VM in terms of CPU power
VMCSIZE:     dynamic array(DOMPU, range)           of real

!---------------------------------------------*
!PARAMETERS TO CALCULATE
!---------------------------------------------*

!The number of replication settings for each service
RMAX:        array(SERV)                           of integer
!Stationary availability of each component using replication pattern r
AVCOMPREAL: dynamic array(SERV, range, range)      of real
!Stationary availability of a service given a replication pattern r
AVAIL:       array(SERV, range)                    of real
!Number of replicas of a comp given a replication pattern r
NR:          dynamic array(SERV, range, range)     of integer
!The maximum number of NR(i,q,r) over r
NRMX:        dynamic array(SERV, range)            of integer
!Number of active replicas of a comp given a replication pattern r
ACTR:        dynamic array(SERV, range, range)     of integer
!The maximum number of ACTR(i,q,r) over r
ACTRMX:      dynamic array(SERV, range)            of integer
!The minimum number of ACTR(i,q,r) over r for components bind to private cloud domain
ACTRMNB:     dynamic array(SERV, range)            of integer
!The total response time of a service given a replication pattern r
RTTOT:       dynamic array(SERV, range)            of real
!The demanded CPU power from requests
CPUDEM:      dynamic array(SERV, range)            of real
!The cost of deploying an active or semi-active replica in a public cloud
COSTVMACT:   dynamic array(SERV, range, DOMPU)     of real
!The cost of deploying a passive replica of (i,q) in a public cloud
COSTVMPAS:   dynamic array(SERV, range, DOMPU)     of real

!Data structure used to predeploy the component which has the highest value of ACTRMN
PREDEP=      record
                  i,q,a:       integer
             end-record
PREDEPARR:  array(DOMPR)                           of PREDEP


!---------------------------------------------*
!VARIABLES
!---------------------------------------------*

!= 1 if a replica (i,q) is deployed in cloud domain, = 0 otherwise, cf. u_iqd
repincl:     dynamic array(SERV, range, DOM)       of mpvar
!The number of replicas of (i,q) that is deployed in a cloud domain, cf. x_Diqd
mappingcl:  dynamic array(SERV, range, DOM)        of mpvar
!= 1 if a replica of (i,q) is deployed on node n in a private cloud domain d, = 0 otherwise, cf x_iqdn
mapping:     dynamic array(SERV, range, DOMPR, range)of mpvar
!The number of active replicas of (i,q) that is deployed in a cloud domain d, cf. w_Diqd
activecl:   dynamic array(SERV, range, DOM)        of mpvar
!= 1 if an active replica of (i,q) is deployed on node n in private cloud domain d, = 0 otherwise, cf. w_iqdn
active:      dynamic array(SERV, range, DOMPR, range)of mpvar
!The number of semi-active replicas of (i,q) deployed in cloud domain d, cf. v_Diqd
semiactcl:  dynamic array(SERV, range, DOM)        of mpvar
!= 1 if a semi-active replicas of (i,q) is deployed on node n in private cloud domain d, = 0 otherwise, cf. v_iqdn
semiact:     dynamic array(SERV, range, DOMPR, range)of mpvar
!The fraction of demand-service replicas of (i,q) that is deployed in private cloud domain d, cf f_iqd
fracact:    dynamic array(SERV, range, DOMPR)      of mpvar
!= 1 if replication pattern r is used for service i, = 0 otherwise, cf. y_ir
repset:      dynamic array(SERV, range)            of mpvar
!= 1 if node n in a private cloud domain is turned on, = 0 otherwise, cf. o_dn
used:        dynamic array(DOMPR, range)           of mpvar
!Amount of non-assignable CPU power on node n in private cloud domain d, cf. m_dn
maxrepcap:  dynamic array(DOMPR, range)            of mpvar
!Amount of CPU power assigned to the replicas on node n in private cloud domain d (used in SYM1), cf. s_dn
assignedc:  dynamic array(DOMPR, range)            of mpvar
```

136

```
    !----------------------------------------------*
    !CONSTRAINTS
    !----------------------------------------------*

    !Ensure that one and only on replication pattern is used for service i
    Repsetsum:  array(SERV)                         of linctr
    !Distribute the number of replicas to the domains
    Allreps:    dynamic array(SERV, range)          of linctr
    !Further map the replicas deployed in private cloud domain d to the nodes
    Allrepsnod: dynamic array(SERV, range, DOMPR)       of linctr
    !Distribute the number of active replicas to the domains
    Allact:     dynamic array(SERV, range)          of linctr
    !Further map the active replicas deployed in private cloud domain d to the nodes
    Allactnod:  dynamic array(SERV, range, DOMPR)       of linctr
    !Distribute the number of semi-active replicas to the domains
    Allsemiact: dynamic array(SERV, range)          of linctr
    !Further map the semi-active replicas deployed in private cloud domain d to the nodes
    Allsactnod: dynamic array(SERV, range, DOMPR)       of linctr
    !Ensure that there are not more active and semi-active replicas than the total number of replicas in a public cloud
    Actif:      dynamic array(SERV, range, DOMPU)       of linctr
    !Ensure that if a replica is active or semi-active in the private cloud it has to be deployed on that node
    Actifnod:   dynamic array(SERV, range, DOMPR, range)    of linctr
    !Ensure the required number of used clouds by the replicas of (i,q)
    Reqspread:  dynamic array(SERV, range)          of linctr
    !Force repincl to 0 if no replicas in the cloud
    Repinclif:  dynamic array(SERV, range, DOM)         of linctr
    !Set fracact variable
    Setfracact: dynamic array(SERV, range, DOMPR, range)    of linctr
    !Set the assignedc variables
    Setass:     dynamic array(DOMPR, range)         of linctr
    !Set the non-assignable amount of CPU power on the nodes
    Setmxrepcap:dynamic array(SERV, range, DOMPR, range)    of linctr
    !CPU capacity constraint on all nodes in private cloud
    Cpucap:     dynamic array(DOMPR, range)         of linctr
    !Sort the nodes according to assignedc
    Sortass:    array(DOMPR, range)                 of linctr
    !Sort the nodes according to used
    Sortused:   array(DOMPR, range)                 of linctr
    !----------------------------------------------*
    !OBJECTIVE
    !----------------------------------------------*
    !Minimize energy costs and public cloud deployment costs
    Obj:                                            linctr

end-declarations


!-------------------------------------------------*
! Initializations
!-------------------------------------------------*
initializations from DATA
    HRS NOS NOCLPR NOCLPU CPWR MODE SYM
end-initializations

SERV    := 1..NOS
DOMPR   := 1..NOCLPR
DOMPU   := (NOCLPR+1)..(NOCLPR+NOCLPU)
DOM     := DOMPR + DOMPU

initializations from DATA
    NOC CAPCPU REQAVAIL REQRESPT PWRCOEFF PWRIDLE DEM VAR NOVM NON
end-initializations

forall(i in SERV) COMP(i) := 1..NOC(i)
forall(d in DOMPU) VMS(d) := 1..NOVM(d)
forall(d in DOMPR) NODES(d) := 1..NON(d)

initializations from DATA
        [MAXREP, MAXACT, AVAILCOMP, SACOEFF, SPREAD, BINDING] AS "REPDATA"
        [JOBLOAD, CPUASS, CPUOH] AS "LOADDATA"
        [COSTVM, VMCSIZE] AS "VMDATA"
end-initializations

!Calculate CPUDEM(i,q)
forall(i in SERV, q in COMP(i)) do
    CPUDEM(i,q) := DEM(i)*JOBLOAD(i,q)
end-do

!Calculate VM cost of stand-by replicas (choose the cheapest one in each cloud)
forall(i in SERV, q in COMP(i), d in DOMPU) do
    COSTVMPAS(i,q,d) := COSTVM(d,1)
end-do
```

```
!Calculate VM cost of active and "semi-active" replicas in each cloud
forall(i in SERV, q in COMP(i), d in DOMPU) do
    !Initialize COSTVMACT(i,q,d) to a large number (larger than the cost of any VM)
    COSTVMACT(i,q,d) := 10000
    CONF := false
    forall(v in VMS(d) | CPUASS(i,q)*VAR(i)+CPUOH(i,q) < VMCSIZE(d,v)) do
        if(COSTVM(d,v) < COSTVMACT(i,q,d)) then
            COSTVMACT(i,q,d) := COSTVM(d,v)
            CONF := true
        end-if
    end-do
    if(not CONF) then
        writeln("Could not find any large enough VMs to fit (",i,",",q,") in ",d)
    end-if
    COSTVMACT(i,q,d) := COSTVMACT(i,q,d)
end-do


!-------------------------------------------------*
! PROCEDURES
!-------------------------------------------------*
forward function doFactorial(x:integer):integer
forward procedure createReplicationPatterns
forward function calcAvail(N:array(SERV, range) of integer, A:array(SERV, range) of integer, i:integer):real
forward function calcRespt(A:array(SERV, range) of integer, i:integer):real
forward function checkStp(TMP:array(SERV, range) of integer, UP:array(SERV,range) of integer, i:integer):boolean
forward function checkAllGeqA(TMPA:array(SERV, range) of integer, i:integer):boolean
forward function checkAllGeqN(TMPN:array(SERV, range) of integer, TMPA:array(SERV, range) of integer, i:integer):boolean


!-------------------------------------------------*
! PROCEDURE TO GENERATE REPLICATION PATTERNS
!-------------------------------------------------*
procedure createReplicationPatterns
    declarations
        !Temporary total number of replicas
        TMPN:   array(SERV, range)  of integer
        !Temporary number of active replicas
        TMPA:   array(SERV, range)  of integer
        !Temporary upper bound on the number of active replicas
        AUP:    array(SERV, range)  of integer
        !Temporary upper bound on the total number of replicas
        NUP:    array(SERV, range)  of integer
        !Temporary lower bound on the number of active replicas
        ALW:    array(SERV, range)  of integer
        !Temporary lower bound on the total number of replicas
        NLW:    array(SERV, range)  of integer
        !Current replication pattern
        r:                          integer
    end-declarations
    !Do forall services
    forall(i in SERV) do
        !Initialize r = 1
        r := 1
        !Initialize the number of active replicas
        forall(q in COMP(i)) do
            TMPA(i,q) := 1
            AUP(i,q) := MAXACT(i,q)
            ALW(i,q) := 1
        end-do
        !FRISTCHCKA used to stop the loop if one needs only 1 active replica of each component
        FIRSTCHCKA := true
        !Outer loop
        while(checkStp(TMPA,AUP,i)) do
            !Check response time requirement and if there exists a clearly better AC already
            if(calcRespt(TMPA,i) <= REQRESPT(i) and checkAllGeqA(TMPA,i)) then
                !Initialize the total number of replicas
                forall(q in COMP(i)) do
                    TMPN(i,q) := TMPA(i,q)
                    NUP(i,q) := MAXREP(i,q)
                    NLW(i,q) := TMPA(i,q)
                end-do
                !Lower temporary upper bound if only one component have another number of active replicas than 1
                AUPCHCK := 0
                forall(q in COMP(i) | TMPA(i,q) <> 1) AUPCHCK += 1
                if(AUPCHCK = 1) then
                    forall(q in COMP(i) | TMPA(i,q) <> 1) AUP(i,q) := TMPA(i,q)
                end-if
                !FRISTCHCKN used to stop the loop if one needs no passive replicas of a component
                FIRSTCHCKN := true
                !Inner loop
                while(checkStp(TMPN,NUP,i)) do
                    !Check availability requirement and if there exists a clearly better TC, based on the same AC
                    if(calcAvail(TMPN,TMPA,i) >= REQAVAIL(i) and checkAllGeqN(TMPN, TMPA, i)) then
                        !Update NR, ACTR, the availabilities, the response time and RSET
                        forall(q in COMP(i)) do
                            NR(i,q,r) := TMPN(i,q)
                            ACTR(i,q,r) := TMPA(i,q)
                        end-do
```

138

```
                    AVAIL(i,r) := calcAvail(TMPN,TMPA,i)
                    RTTOT(i,r) := calcRespt(TMPA,i)
                    RSET(i) += {r}
                    RMAX(i) := r
                    !Increment r
                    r += 1
                    !Break loop if FRISTCHCKN is true
                    if(FIRSTCHCKN) then
                        break
                    else
                        !Lower temporary upper bound if only one component have another # passive replicas than 1
                        NUPCHCK := 0
                        forall(q in COMP(i) | TMPN(i,q) <> TMPA(i,q)) NUPCHCK += 1
                        if(NUPCHCK = 1) then
                            forall(q in COMP(i) | TMPN(i,q) <> TMPA(i,q)) NUP(i,q) := TMPN(i,q)
                        end-if
                    end-if
                end-if !REQAVAIL
                FIRSTCHCKN := false
                !Find new TMPN(i)
                q := NOC(i)
                while(q > 1) do
                    qq := NOC(i)
                    while(qq > 0) do
                        if(TMPN(i,qq) < NUP(i,qq)) then
                            TMPN(i,qq) += 1
                            break 2
                        else
                            TMPN(i,qq) := NLW(i,qq)
                            if(qq = 1) then
                                NLW(i,q) += 1
                                TMPN(i,q) := NLW(i,q)
                                if(q = 1) then
                                    break 2
                                else
                                    qq := NOC(i) - q
                                    q -= 1
                                end-if
                            else
                                qq -= 1
                            end-if
                        end-if
                        !qq -= 1
                    end-do  !qq
                end-do !q
            end-do !checkStp(TMPN,NUP,i)
            if(FIRSTCHCKA) then
                break
            end-if
        end-if !REQRESPT
        FIRSTCHCKA := false

        !Find new TMPA(i)
        q := NOC(i)
        while(q > 1) do
            qq := NOC(i)
            while(qq > 0) do
                if(TMPA(i,qq) < AUP(i,qq)) then
                    TMPA(i,qq) += 1
                    break 2
                else
                    TMPA(i,qq) := ALW(i,qq)
                    if(qq = 1) then
                        !NLW(i,qq) += 1
                        ALW(i,q) += 1
                        TMPA(i,q) := ALW(i,q)
                        if(q = 1) then
                            break 2
                        else
                            qq := NOC(i) - q
                            q -= 1
                        end-if
                    else
                        qq -= 1
                    end-if
                end-if
            end-do  !qq
        end-do !q
    end-do !checkStp(TMPA,AUP,i)
    end-do !forall i
end-procedure
```

```
!Calculates the response time based on TMPA for i
function calcRespt(A:array(SERV, range) of integer, i:integer):real
    !Check if stable queue
    INF := false
    forall(q in COMP(i)) do
        if(CPUASS(i,q)/JOBLOAD(i,q) - DEM(i)/A(i,q) <= 0) then
            INF := true
        end-if
    end-do
    if(INF = false) then
        !Multiply be 1000 to get milliseconds
        returned := sum(q in COMP(i)) 1000/(CPUASS(i,q)/JOBLOAD(i,q) - DEM(i)/A(i,q))
    else
        !Else return a very high number
        returned := 1000000000
    end-if
end-function

!Calculate the service availability based on TMPA and TMPN for i
function calcAvail(N:array(SERV, range) of integer, A:array(SERV, range) of integer, i:integer):real
    returned := prod(q in COMP(i)) sum(k in A(i,q)..N(i,q))(doFactorial(N(i,q))/(doFactorial(k)*doFactorial(N(i,q)-k)))*
                                            AVAILCOMP(i,q)^k*(1-AVAILCOMP(i,q))^(N(i,q)-k)
end-function

!Chech if loop should stop
function checkStp(TMP:array(SERV, range) of integer, UP:array(SERV,range) of integer, i:integer):boolean
    retval := false
    forall(q in COMP(i) | TMP(i,q) <> UP(i,q)) retval := true
    returned := retval
end-function

!Check if there exists a clearly better AC already
function checkAllGeqA(TMPA:array(SERV, range) of integer, i:integer):boolean
    retval := true
    forall(r in RSET(i)) do
        check := 0
        forall(q in COMP(i)) do
            if(TMPA(i,q) >= ACTR(i,q,r)) then
                check += 1
            end-if
        end-do
        if(check = NOC(i)) then
            retval := false
            break
        end-if
    end-do
    returned := retval
end-function




!Check if there exists a clearly better TC, based on the same AC
function checkAllGeqN(TMPN:array(SERV, range) of integer, TMPA:array(SERV, range) of integer, i:integer):boolean
    declarations
        !Set of replication patterns that is based on the same AC, cf. R*
        TMPRSET:set of integer
    end-declarations
    retval := true
    !Calculate R*
    forall(r in RSET(i)) do
        checkr := 0
        forall(q in COMP(i)) do
            if(TMPA(i,q) = ACTR(i,q,r)) then
                checkr += 1
            end-if
        end-do
        if(checkr = NOC(i)) then
            TMPRSET += {r}
        end-if
    end-do
    forall(r in TMPRSET) do
        checkn := 0
        forall(q in COMP(i)) do
            if(TMPN(i,q) >= NR(i,q,r)) then
                checkn += 1
            end-if
        end-do
        if(checkn = NOC(i)) then
            retval := false
            break
        end-if
    end-do
    returned := retval
end-function
```

140

```
!doFacorial - Function which computes the factorial of an integer
function doFactorial(x:integer):integer
        if (x <= 1) then
            returned:=1;
        else
            returned:=(x * doFactorial(x-1));
        end-if
end-function

writeln("createReplicationPatterns BEGIN")
createReplicationPatterns
writeln("createReplicationPatterns DONE")


!Calculate ACTRMX(i,q)
forall(i in SERV, q in COMP(i)) do
    ACTRMX(i,q) := 1
    forall(r in RSET(i) | ACTR(i,q,r) > ACTRMX(i,q)) ACTRMX(i,q) := ACTR(i,q,r)
end-do

!Calculate ACTRMNB(i,q)
forall(i in SERV, q in COMP(i) | BINDING(i,q) in DOMPR) do
    ACTRMNB(i,q) := MAXACT(i,q)
    forall(r in RSET(i) | ACTR(i,q,r) < ACTRMNB(i,q)) ACTRMNB(i,q) := ACTR(i,q,r)
end-do

!Calculate NRMX(i,q)
forall(i in SERV, q in COMP(i)) do
    NRMX(i,q) := 1
    forall(r in RSET(i) | NR(i,q,r) > NRMX(i,q)) NRMX(i,q) := NR(i,q,r)
end-do

!Print out replication settings
forall(i in SERV) do
    writeln("--------SERVICE ",i," -------------")
    write("r ")
    forall(q in COMP(i))do
        write("a_",q," n_",q,"     ")
    end-do
    writeln("AVAIL(",i,",r)                      RTTOT(",i,",r)")
    forall(r in RSET(i)) do
        write(r," ")
        forall(q in COMP(i)) do
            write(" ",ACTR(i,q,r),"   ",NR(i,q,r),"     ")
        end-do
        writeln(AVAIL(i,r)," >= ",REQAVAIL(i),"        ",RTTOT(i,r)," <= ",REQRESPT(i))
    end-do
end-do




!-------------------------------------------------*
!CREATE VARIABLES
!-------------------------------------------------*
forall(i in SERV, q in COMP(i)) do
    if(BINDING(i,q)=0) then
        forall(d in DOM) do
            !mappingcl(i,q,d)
            create(mappingcl(i,q,d))
            mappingcl(i,q,d) is_integer
            !activecl(i,q,d)
            create(activecl(i,q,d))
            activecl(i,q,d) is_integer
            if(MODE = 1) then
                !semiactcl(i,q,d)
                create(semiactcl(i,q,d))
                semiactcl(i,q,d) is_integer
            end-if
            !repincl(i,q,d)
            create(repincl(i,q,d))
            repincl(i,q,d) is_binary
            if(d in DOMPR) then
                !fracact(i,q,d)
                create(fracact(i,q,d))
            end-if
        end-do
    else
        !mappingcl(i,q,BINDING(i,q))
        create(mappingcl(i,q,BINDING(i,q)))
        mappingcl(i,q,BINDING(i,q)) is_integer
        !activecl(i,q,BINDING(i,q))
        create(activecl(i,q,BINDING(i,q)))
        activecl(i,q,BINDING(i,q)) is_integer
        if(MODE = 1) then
            !semiactcl(i,q,BINDING(i,q))
            create(semiactcl(i,q,BINDING(i,q)))
            semiactcl(i,q,BINDING(i,q)) is_integer
        end-if
```

141

```
        !repincl(i,q,BINDING(i,q))
        create(repincl(i,q,BINDING(i,q)))
        repincl(i,q,BINDING(i,q)) is_binary
        if(BINDING(i,q) in DOMPR) then
            !fracact(i,q,BINDING(i,q))
            create(fracact(i,q,BINDING(i,q)))
        end-if
    end-if
end-do


forall(i in SERV, q in COMP(i)) do!, d in DOMPR, n in NODES(d) | BINDING(i,q) = 0 or BINDING(i,q) = d) do
    if(BINDING(i,q) = 0) then
        forall(d in DOMPR, n in NODES(d)) do
            !mapping(i,q,d,n)
            create(mapping(i,q,d,n))
            mapping(i,q,d,n) is_binary
            if(MODE <> 2) then
                !active(i,q,d,n)
                create(active(i,q,d,n))
                active(i,q,d,n) is_binary
                if(MODE = 1) then
                    !semiact(i,q,d,n)
                    create(semiact(i,q,d,n))
                    semiact(i,q,d,n) is_binary
                end-if
            end-if
        end-do
    else
        if(BINDING(i,q) in DOMPR) then
            forall(n in NODES(BINDING(i,q))) do
                !mapping(i,q,BINDING(i,q),n)
                create(mapping(i,q,BINDING(i,q),n))
                mapping(i,q,BINDING(i,q),n) is_binary
                if(MODE <> 2) then
                    !active(i,q,BINDING(i,q),n)
                    create(active(i,q,BINDING(i,q),n))
                    active(i,q,BINDING(i,q),n) is_binary
                    if(MODE = 1) then
                        !semiact(i,q,BINDING(i,q),n)
                        create(semiact(i,q,BINDING(i,q),n))
                        semiact(i,q,BINDING(i,q),n) is_binary
                    end-if
                end-if
            end-do
        end-if
    end-if
end-do

!repset(i,r)
forall(i in SERV, r in RSET(i)) do
    create(repset(i,r))
    repset(i,r) is_binary
end-do

forall(d in DOMPR, n in NODES(d)) do
    !used(d,n)
    create(used(d,n))
    used(d,n) is_binary
    !assignedc(d,n)
    create(assignedc(d,n))
    if(MODE = 0) then
        !maxrepcap(d,n)
        create(maxrepcap(d,n))
    end-if
end-do


if(MODE = 0) then
    !********************************************************************************************
    !BEGIN CAPACITY LOWERING APPROACH
    !********************************************************************************************

    !------------------------------------------------*
    ! DEFINE OBJECTIVE FUNCTION
    !------------------------------------------------*

    Obj :=    CPWR*HRS*(sum(i in SERV, q in COMP(i), d in DOMPR)PWRCOEFF*CPUDEM(i,q) * fracact(i,q,d) +
          sum(i in SERV, q in COMP(i), d in DOMPR, n in NODES(d))PWRCOEFF*CPUOH(i,q) * mapping(i,q,d,n) +
                                   sum(d in DOMPR, n in NODES(d))PWRIDLE * used(d,n)) +
                    HRS*sum(i in SERV, q in COMP(i), d in DOMPU)COSTVMACT(i,q,d) * activecl(i,q,d) +
                    HRS*sum(i in SERV, q in COMP(i), d in DOMPU)COSTVMPAS(i,q,d) * (mappingcl(i,q,d)-
activecl(i,q,d))
```

```
!------------------------------------------------*
! CONSTRAINTS
!------------------------------------------------*

!Choose one and only one replication pattern for each service
forall(i in SERV) Repsetsum(i) := sum(r in RSET(i)) repset(i,r) = 1

!Distribute the replicas to the domains
forall(i in SERV, q in COMP(i)) Allreps(i,q) := sum(d in DOM)                mappingcl(i,q,d) -
                                        sum(r in RSET(i)) NR(i,q,r) * repset(i,r) = 0

!Further map replicas in the private domains to the nodes
forall(i in SERV, q in COMP(i), d in DOMPR) do
    Allrepsnod(i,q,d) := sum(n in NODES(d)) mapping(i,q,d,n) -
                                        mappingcl(i,q,d) = 0
end-do

!Distribute the active replicas to the domains
forall(i in SERV, q in COMP(i)) Allact(i,q) := sum(d in DOM)                activecl(i,q,d) -
                                        sum(r in RSET(i)) ACTR(i,q,r) * repset(i,r) = 0

!Further map replicas in the private domains to the nodes
forall(i in SERV, q in COMP(i), d in DOMPR) do
    Allactnod(i,q,d) := sum(n in NODES(d)) active(i,q,d,n) -
                                        activecl(i,q,d) = 0
end-do

!Set the fraction of demand-serving replicas in the private cloud
forall(i in SERV, q in COMP(i), d in DOMPR, r in RSET(i)) do
    Setfracact(i,q,d,r) :=              1/ACTR(i,q,r) * activecl(i,q,d) -
                                                fracact(i,q,d) +
                        ACTRMX(i,q)/ACTR(i,q,r) * repset(i,r) <= ACTRMX(i,q)/ACTR(i,q,r)
end-do

!Ensure that the number of activated replicas are less than or equal to the total number of replicas
forall(i in SERV, q in COMP(i), d in DOMPU) Actif(i,q,d) := activecl(i,q,d) -
                                                mappingcl(i,q,d) <= 0

!Ensure that if a replica is activated on a node it has to be deployed there
forall(i in SERV, q in COMP(i), d in DOMPR, n in NODES(d)) Actifnod(i,q,d,n) := active(i,q,d,n) -
                                                        mapping(i,q,d,n) <= 0

!Ensure the minimum number of domains used
forall(i in SERV, q in COMP(i) | BINDING(i,q) = 0) do
    Reqspread(i,q) :=                        sum(d in DOM)   repincl(i,q,d) -
                    sum(r in RSET(i)) NR(i,q,r)*SPREAD(i,q) * repset(i,r) >= 0
end-do

!Force repincl to be 0 if there is not deployed any replicas in the domain
forall(i in SERV, q in COMP(i), d in DOM) Repinclif(i,q,d) := mappingcl(i,q,d) -
                                                repincl(i,q,d) >= 0

!Set the non-assignable amount of CPU power on the nodes in the private cloud domains
forall(d in DOMPR, n in NODES(d), i in SERV, q in COMP(i)) do
    Setmxrepcap(i,q,d,n) :=                      maxrepcap(d,n) -
                        (CPUASS(i,q)*VAR(i)) * (mapping(i,q,d,n) - active(i,q,d,n)) >= 0
end-do

if(SYM = 1) then
    !Set the amount of CPU power assigned to the replicas on the nodes in the private domains
    forall(d in DOMPR, n in NODES(d)) do
        Setass(d,n) := sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i)) * active(i,q,d,n) +
                                sum(i in SERV, q in COMP(i))CPUOH(i,q) * mapping(i,q,d,n) -
                                                        assignedc(d,n) = 0
    end-do

    !Capacity constraints on the nodes in the private domains
    forall(d in DOMPR, n in NODES(d)) do
        Cpucap(d,n) :=              assignedc(d,n) +
                                maxrepcap(d,n) -
                    CAPCPU * used(d,n) <= 0
    end-do
else
    !Capacity constraints on the nodes in the private domains
    forall(d in DOMPR, n in NODES(d)) do
        Cpucap(d,n) := sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i)) * active(i,q,d,n) +
                                sum(i in SERV, q in COMP(i))CPUOH(i,q) * mapping(i,q,d,n) +
                                                        maxrepcap(d,n) -
                                                CAPCPU * used(d,n) <= 0
    end-do
end-if


!****************************************************************************************************
!END CAPACITY LOWERING APPROACH
!****************************************************************************************************
```

143

```
elif(MODE = 1) then
    !***********************************************************************************************
    !BEGIN SEMI-ACTIVE APPROACH
    !***********************************************************************************************

    !-------------------------------------------------*
    ! DEFINE OBJECTIVE FUNCTION
    !-------------------------------------------------*

    Obj :=      CPWR*HRS*(sum(i in SERV, q in COMP(i), d in DOMPR)PWRCOEFF*CPUDEM(i,q) * fracact(i,q,d) +
            sum(i in SERV, q in COMP(i), d in DOMPR, n in NODES(d))PWRCOEFF*CPUOH(i,q) * mapping(i,q,d,n) +
                                    sum(d in DOMPR, n in NODES(d))PWRIDLE * used(d,n)) +
                    HRS*sum(i in SERV, q in COMP(i), d in DOMPU)COSTVMACT(i,q,d) * activecl(i,q,d) +
                    HRS*sum(i in SERV, q in COMP(i), d in DOMPU)COSTVMACT(i,q,d) * semiactcl(i,q,d) +
                    HRS*sum(i in SERV, q in COMP(i), d in DOMPU)COSTVMPAS(i,q,d) * (mappingcl(i,q,d)-
                                                                                  (activecl(i,q,d)+
                                                                                   semiactcl(i,q,d)))

    !-------------------------------------------------*
    ! CONSTRAINTS
    !-------------------------------------------------*

    !Choose one and only one replication pattern for each service
    forall(i in SERV) Repsetsum(i) := sum(r in RSET(i)) repset(i,r) = 1

    !Distribute the replicas to the domains
    forall(i in SERV, q in COMP(i)) Allreps(i,q) := sum(d in DOM)          mappingcl(i,q,d) -
                                        sum(r in RSET(i)) NR(i,q,r) * repset(i,r) = 0

    !Further map replicas in the private domains to the nodes
    forall(i in SERV, q in COMP(i), d in DOMPR) do
        Allrepsnod(i,q,d) := sum(n in NODES(d)) mapping(i,q,d,n) -
                                        mappingcl(i,q,d) = 0
    end-do

    !Distribute the active replicas to the domains
    forall(i in SERV, q in COMP(i)) Allact(i,q) := sum(d in DOM)            activecl(i,q,d) -
                                        sum(r in RSET(i)) ACTR(i,q,r) * repset(i,r) = 0

    !Further map active replicas in the private domains to the nodes
    forall(i in SERV, q in COMP(i), d in DOMPR) do
        Allactnod(i,q,d) := sum(n in NODES(d)) active(i,q,d,n) -
                                        activecl(i,q,d) = 0
    end-do

    !Distribute the semi-active replicas to the domains
    forall(i in SERV, q in COMP(i)) do
        Allsemiact(i,q) :=                                      sum(d in DOM)   semiactcl(i,q,d) -
                    sum(r in RSET(i))ceil((NR(i,q,r)-ACTR(i,q,r))/ACTR(i,q,r)) * repset(i,r) = 0
    end-do

    !Further map semi-active replicas in the private domains to the nodes
    forall(i in SERV, q in COMP(i), d in DOMPR) do
        Allsactnod(i,q,d) := sum(n in NODES(d)) semiact(i,q,d,n) -
                                        semiactcl(i,q,d) = 0
    end-do

    !Ensure that the number of activated and semi-active replicas are less than or equal to the total number of replicas
    forall(i in SERV, q in COMP(i), d in DOMPU) Actif(i,q,d) :=     activecl(i,q,d) +
                                                    semiactcl(i,q,d) -
                                                    mappingcl(i,q,d) <= 0


    !Ensure that if a replica is activated or semi-active on a node it has to be deployed there
    forall(i in SERV, q in COMP(i), d in DOMPR, n in NODES(d)) Actifnod(i,q,d,n) := active(i,q,d,n) +
                                                    semiact(i,q,d,n) -
                                                    mapping(i,q,d,n) <= 0

    !Set the fraction of demand-serving replicas in the private domain
    forall(i in SERV, q in COMP(i), d in DOMPR, r in RSET(i)) do
        Setfracact(i,q,d,r) :=          1/ACTR(i,q,r) * activecl(i,q,d) -
                                                    fracact(i,q,d) +
                        ACTRMX(i,q)/ACTR(i,q,r) * repset(i,r) <= ACTRMX(i,q)/ACTR(i,q,r)
    end-do

    !Ensure the minimum number of domains used
    forall(i in SERV, q in COMP(i) | BINDING(i,q) = 0) do
        Reqspread(i,q) :=                           sum(d in DOM)   repincl(i,q,d) -
                    sum(r in RSET(i)) NR(i,q,r)*SPREAD(i,q) * repset(i,r) >= 0
    end-do

    !Force repincl to be 0 if there is not deployed any replicas in the domain
    forall(i in SERV, q in COMP(i), d in DOM) Repinclif(i,q,d) := mappingcl(i,q,d) -
                                                    repincl(i,q,d) >= 0
```

144

```
      if(SYM = 1) then
          !Set the amount of CPU power assigned to the replicas on the nodes in the private domains
          forall(d in DOMPR, n in NODES(d)) do
          Setass(d,n) := sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i)) * active(i,q,d,n) +
                         sum(i in SERV, q in COMP(i))CPUASS(i,q)*VAR(i) * semiact(i,q,d,n) +
                                sum(i in SERV, q in COMP(i))CPUOH(i,q) * mapping(i,q,d,n) -
                                                                assignedc(d,n) = 0
          end-do

          !Capacity constraint on the nodes in the private domains
          forall(d in DOMPR, n in NODES(d)) do
              Cpucap(d,n) :=              assignedc(d,n) -
                         CAPCPU * used(d,n) <= 0
          end-do
      else
          !Capacity constraint on the nodes in the private domains
          forall(d in DOMPR, n in NODES(d)) do
              Cpucap(d,n) := sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i)) * active(i,q,d,n) +
                            sum(i in SERV, q in COMP(i))CPUASS(i,q)*VAR(i) * semiact(i,q,d,n) +
                                   sum(i in SERV, q in COMP(i))CPUOH(i,q) * mapping(i,q,d,n) -
                                                                   CAPCPU * used(d,n) <= 0
          end-do
      end-if

      !********************************************************************************************
      !END SEMI-ACTIVE APPROACH
      !********************************************************************************************

elif(MODE = 2) then

      !********************************************************************************************
      !BEGIN ALL-ACTIVE APPROACH
      !********************************************************************************************

      !-------------------------------------------------*
      ! DEFINE OBJECTIVE FUNCTION
      !-------------------------------------------------*

      Obj :=      CPWR*HRS*(sum(i in SERV, q in COMP(i), d in DOMPR)PWRCOEFF*CPUDEM(i,q) * fracact(i,q,d) +
              sum(i in SERV, q in COMP(i), d in DOMPR, n in NODES(d))PWRCOEFF*CPUOH(i,q) * mapping(i,q,d,n) +
                                   sum(d in DOMPR, n in NODES(d))PWRIDLE * used(d,n)) +
                       HRS*sum(i in SERV, q in COMP(i), d in DOMPU)COSTVMACT(i,q,d) * mappingcl(i,q,d)


      !-------------------------------------------------*
      ! CONSTRAINTS
      !-------------------------------------------------*

      !Choose one and only one replication pattern for each service
      forall(i in SERV) Repsetsum(i) := sum(r in RSET(i)) repset(i,r) = 1

      !Distribute the replicas to the domains
      forall(i in SERV, q in COMP(i)) Allreps(i,q) := sum(d in DOM)                mappingcl(i,q,d) -
                                          sum(r in RSET(i)) NR(i,q,r) * repset(i,r) = 0

      !Further map replicas in the private domains to the nodes
      forall(i in SERV, q in COMP(i), d in DOMPR) do
          Allrepsnod(i,q,d) := sum(n in NODES(d)) mapping(i,q,d,n) -
                                          mappingcl(i,q,d) = 0
      end-do



      !Set the fraction of demand-serving replicas in the private domain
      forall(i in SERV, q in COMP(i), d in DOMPR, r in RSET(i)) do
          Setfracact(i,q,d,r) :=          1/NR(i,q,r) * mappingcl(i,q,d) -
                                               fracact(i,q,d) +
                             NRMX(i,q)/NR(i,q,r) * repset(i,r) <= NRMX(i,q)/NR(i,q,r)
      end-do

      !Ensure the minimum number of domains used
      forall(i in SERV, q in COMP(i) | BINDING(i,q) = 0) do
          Reqspread(i,q) :=                         sum(d in DOM)  repincl(i,q,d) -
                         sum(r in RSET(i)) NR(i,q,r)*SPREAD(i,q) * repset(i,r) >= 0
      end-do

      !Force repincl to be 0 if there is not deployed any replicas in the domain
      forall(i in SERV, q in COMP(i), d in DOM) Repinclif(i,q,d) := mappingcl(i,q,d) -
                                                        repincl(i,q,d) >= 0


      if(SYM = 1) then
          !Set the amount of CPU power assigned to the replicas on the nodes in the private domains
          forall(d in DOMPR, n in NODES(d)) do
              Setass(d,n) := sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i)+CPUOH(i,q)) * mapping(i,q,d,n) -
                                                              assignedc(d,n) = 0
          end-do
```

```
        !Capacity constraints on the nodes in the private domain
        forall(d in DOMPR, n in NODES(d)) do
            Cpucap(d,n) :=              assignedc(d,n) -
                        CAPCPU * used(d,n) <= 0
        end-do
    else
        !Capacity constraints on the nodes in the private domain
        forall(d in DOMPR, n in NODES(d)) do
            Cpucap(d,n) := sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i)+CPUOH(i,q)) * mapping(i,q,d,n) -
                                                        CAPCPU * used(d,n) <= 0
        end-do
    end-if

    !***********************************************************************************************
    !END ALL-ACTIVE APPROACH
    !***********************************************************************************************
end-if

!Doing some predefined decisions in order to lower the number of symmetrical solutions
!The component with the highest ACTRMN is deployed on the nodes with the lowest index

if(SYM = 1) then
    forall(d in DOMPR, n in NODES(d) | n < NON(d)) Sortass(d,n) := assignedc(d,n) - assignedc(d,n+1) >= 0
elif(SYM = 2) then
    forall(d in DOMPR, n in NODES(d) | n < NON(d)) Sortused(d,n) := used(d,n) - used(d,n+1) >= 0
end-if

if(SYM <> 1) then
    !Initialize PREDEPARR(d) for each private cloud domain
    forall(d in DOMPR) do
        forall(i in SERV, q in COMP(i) | BINDING(i,q) = d) do
            PREDEPARR(d).i := i
            PREDEPARR(d).q := q
            PREDEPARR(d).a := ACTRMNB(i,q)
            break
        end-do
    end-do

    !For each private cloud domain: find the component with the highest ACTRMNB that is bound to the cloud domain
    forall(d in DOMPR) do
        forall(i in SERV, q in COMP(i) | PREDEPARR(d).a < ACTRMNB(i,q)) do
            PREDEPARR(d).i := i
            PREDEPARR(d).q := q
            PREDEPARR(d).a := ACTRMNB(i,q)
        end-do
    end-do

    forall(d in DOMPR, n in NODES(d) | n <= PREDEPARR(d).a) do
        if(MODE <> 2) then
            active(PREDEPARR(d).i, PREDEPARR(d).q, d, n) = 1
            writeln("an active replica of (",PREDEPARR(d).i,",",PREDEPARR(d).q,") is predeployed to (",d,",",n,")")
        else
            mapping(PREDEPARR(d).i, PREDEPARR(d).q, d, n) = 1
            writeln("an active replica of (",PREDEPARR(d).i,",",PREDEPARR(d).q,") is predeployed to (",d,",",n,")")
        end-if
    end-do
end-if

writeln("Begin running model")
!--------------------------------------------------*
!MINIMIZE OBJECTIVE
!--------------------------------------------------*
setparam('xprs_verbose',true)
setparam('xprs_miplog', -1000)
setparam('xprs_maxtime',-7000)

minimize(Obj)
writeln("Objective = ",getobjval)
writeln(sum(i in SERV, q in COMP(i), d in DOMPR)CPWR*PWRCOEFF*CPUDEM(i,q) * fracact(i,q,d).sol)
writeln(sum(i in SERV, q in COMP(i), d in DOMPR, n in NODES(d))CPWR*PWRCOEFF*CPUOH(i,q) * mapping(i,q,d,n).sol)
writeln(sum(d in DOMPR, n in NODES(d))CPWR*PWRIDLE * used(d,n).sol)
writeln(sum(i in SERV, q in COMP(i), d in DOMPU)COSTVMACT(i,q,d) * activecl(i,q,d).sol)
writeln(sum(i in SERV, q in COMP(i), d in DOMPU)COSTVMACT(i,q,d) * semiactcl(i,q,d).sol)
writeln(sum(i in SERV, q in COMP(i), d in DOMPU) COSTVMPAS(i,q,d) * (mappingcl(i,q,d).sol-activecl(i,q,d).sol -
semiactcl(i,q,d).sol))

!Print out the number of replciations settings for each service
forall(i in SERV) do
    if(RMAX(i) > 0) then
        writeln("Service ",i," has ",RMAX(i)," replication settings that confins with the SLA requirements")
    else
        writeln("Service ",i," has none replication settings: THE MODEL IS INFEASIBLE!")
    end-if
end-do

!Print out information about availability
writeln("")
writeln("Availability print out")
```

146

```
forall(i in SERV, r in RSET(i) | repset(i,r).sol > 0.1) do
    writeln("-------- SERVICE ",i,"----------")
    writeln("Replication setting ",r," is used")
    writeln("AVAIL(",i,",",r,") = ",AVAIL(i,r)," >= ",REQAVAIL(i)," = REQAVAIL(",i,")")
    forall(q in COMP(i)) do
        writeln("NR(",i,",",q,",",r,") = ",NR(i,q,r))
        writeln("ACTR(",i,",",q,",",r,") = ",ACTR(i,q,r))
    end-do
end-do

!Print out CPULOAD data
writeln("")
writeln("CPULOAD(i,q) and VAR(i) print out")
forall(i in SERV) do
    writeln("-------- SERVICE ",i,"----------")
    writeln("Demand for service ",i,": ",DEM(i), " req/sec")
    writeln("VAR(",i,") = ",VAR(i))
    forall(q in COMP(i)) do
        writeln("*Component ",q,"*")
        writeln("JOBLOAD(",i,",",q,") = ",JOBLOAD(i,q))
        writeln("CPUDEM(",i,",",q,") = ",CPUDEM(i,q))
        writeln("CPUASS(",i,",",q,") = ",CPUASS(i,q))
        writeln("Assigned CPU power including VAR(",i,") to each active replica: ",CPUASS(i,q)*VAR(i))
        writeln("CPUASS(",i,",",q,") * Active reps = ",CPUASS(i,q)," * ",sum(r in RSET(i))ACTR(i,q,r)*repset(i,r).sol,
            " = ",CPUASS(i,q) * sum(r in RSET(i))ACTR(i,q,r)*repset(i,r).sol)
        writeln("This gives a component response time of ",
            1000/(CPUASS(i,q)/JOBLOAD(i,q) - DEM(i)/(sum(r in RSET(i))ACTR(i,q,r)*repset(i,r).sol))," ms")
    end-do
end-do

!Print out information about deployment decisions
writeln("")
writeln("Deployment variables on DOMAIN LEVEL, mappingcl(i,q,d), active(i,q,d) and semiactcl(i,q,d) print out")
forall(i in SERV, q in COMP(i), d in DOM | mappingcl(i,q,d).sol > 0.1) do
    write("# replicas for (",i,",",q,") in cloud ",d," = ",mappingcl(i,q,d).sol," ")
    if(activecl(i,q,d).sol > 0) then
        write("(# active = ",activecl(i,q,d).sol,") ")
    end-if
    if(semiactcl(i,q,d).sol > 0) then
        write("(# semi-active = ",semiactcl(i,q,d).sol,") ")
    end-if
    writeln("")
end-do
writeln("")
writeln("Deployment variables on NODE LEVEL, mapping(i,q,d,n), active(i,q,d,n) and semiact(i,q,d,n) print out")
forall(i in SERV, q in COMP(i), d in DOMPR, n in NODES(d) | mapping(i,q,d,n).sol > 0.1) do
    write("mapping(",i,",",q,",",d,",",n,") = ",mapping(i,q,d,n).sol," ")
    if(active(i,q,d,n).sol > 0.5) then
        write("(active)")
    end-if
    if(semiact(i,q,d,n).sol > 0.5) then
        write("(semiactive)")
    end-if
    writeln("")
end-do

if(MODE <> 2) then
    !Print out information about CPU capacity and power usage on the nodes
    writeln("")
    writeln("Capacity and power usage print out")
    forall(d in DOMPR) do
        writeln("-------------DOMAIN ",d," -------------")
        forall(n in NODES(d)) do
            write("-------- NODE ",n," ")
            if (used(d,n).sol > 0.5) then
                write("(ON) ")
            else
                write("(OFF) ")
            end-if
            writeln("---------")
            forall(i in SERV, q in COMP(i) | mapping(i,q,d,n).sol>0.5) do
                write("(",i,",",q,") with CPU load ",CPUASS(i,q)*VAR(i)*(active(i,q,d,n).sol + semiact(i,q,d,n).sol) +
                    CPUOH(i,q)*mapping(i,q,d,n).sol," ")
                if(active(i,q,d,n).sol > 0.5) then
                    write("(active)")
                end-if
                if(semiact(i,q,d,n).sol > 0.5) then
                    write("(semiactive)")
                end-if
                writeln("")
            end-do
            write("Total load on node ",n,": ",
                sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i)*(active(i,q,d,n).sol + semiact(i,q,d,n).sol) +
                CPUOH(i,q)*mapping(i,q,d,n).sol))
            if(MODE = 0) then
                writeln(" <= CAPCPU - maxrepcap(",n,") = ",CAPCPU,"-",maxrepcap(d,n).sol," = ",CAPCPU-maxrepcap(d,n).sol)
            elif(MODE = 1) then
                writeln(" <= CAPCPU = ",CAPCPU)
```

```
                    end-if
              end-do
          writeln("Total cost of power usage in private cloud = ",
                    CPWR*(sum(i in SERV, q in COMP(i))PWRCOEFF*CPUDEM(i,q)*fracact(i,q,d).sol +
                    sum(i in SERV, q in COMP(i), n in NODES(d))PWRCOEFF*CPUOH(i,q)*mapping(i,q,d,n).sol +
                    sum(n in NODES(d))PWRIDLE*used(d,n).sol))
        end-do
else
    !Print out information about CPU capacity and power usage on the nodes
    writeln("")
    writeln("Capacity and power usage print out")
    forall(d in DOMPR) do
        writeln("------------- DOMAIN ",d," -------------")
        forall(n in NODES(d)) do
            write("--------- NODE ",n," ")
            if (used(d,n).sol > 0.5) then
                write("(ON) ")
            else
                write("(OFF) ")
            end-if
            writeln("---------")
            forall(i in SERV, q in COMP(i) | mapping(i,q,d,n).sol>0.5) do
                writeln("(",i,",",q,") with CPU load ",(CPUASS(i,q)*VAR(i)+CPUOH(i,q))*mapping(i,q,d,n).sol)
            end-do
            writeln("Total load on node ",n,": ",
                    sum(i in SERV, q in COMP(i))(CPUASS(i,q)*VAR(i)+CPUOH(i,q))*mapping(i,q,d,n).sol," <= ",CAPCPU)
        end-do
        writeln("Total cost of power usage in private cloud = ",
                    CPWR*(sum(i in SERV, q in COMP(i))PWRCOEFF*CPUDEM(i,q)*fracact(i,q,d).sol +
                    sum(i in SERV, q in COMP(i), n in NODES(d))PWRCOEFF*CPUOH(i,q)*mapping(i,q,d,n).sol +
                    sum(n in NODES(d))PWRIDLE*used(d,n).sol))
    end-do
end-if

!Print out information about utilization of the nodes in the private cloud
if(MODE <> 2) then
    writeln("")
    forall(d in DOMPR, n in NODES(d)) do
        writeln("")
        forall(i in SERV, q in COMP(i) | mapping(i,q,d,n).sol > 0.1) do
            writeln("(",i,",",q,")'s contribution to the utilization on node ",n," (in ",d,") is: ",
                    CPUOH(i,q)*mapping(i,q,d,n).sol +
                    CPUDEM(i,q)*sum(r in RSET(i))active(i,q,d,n).sol/ACTR(i,q,r)*repset(i,r).sol)
        end-do
        writeln("The total utilization on node ",n," is: ",
                    sum(i in SERV, q in COMP(i))CPUDEM(i,q)*sum(r in RSET(i))active(i,q,d,n).sol/ACTR(i,q,r)*repset(i,r).sol +
                    sum(i in SERV, q in COMP(i))CPUOH(i,q)*mapping(i,q,d,n).sol)
    end-do
    writeln("The average utilization on turned on nodes = ",(sum(i in SERV,q in COMP(i),d in DOMPR,n in NODES(d))
            CPUDEM(i,q)*sum(r in RSET(i))active(i,q,d,n).sol/ACTR(i,q,r)*repset(i,r).sol +
            sum(i in SERV, q in COMP(i), d in DOMPR, n in NODES(d))CPUOH(i,q)*mapping(i,q,d,n).sol)/
            (sum(d in DOMPR, n in NODES(d))used(d,n).sol))
else
    writeln("")
    forall(d in DOMPR, n in NODES(d)) do
        writeln("")
        forall(i in SERV, q in COMP(i) | mapping(i,q,d,n).sol > 0.1) do
            writeln("(",i,",",q,")'s contribution to the utilization on node ",n," (in ",d,") is: ",
                    (CPUOH(i,q))*mapping(i,q,d,n).sol +
                    CPUDEM(i,q)*sum(r in RSET(i))mapping(i,q,d,n).sol/NR(i,q,r)*repset(i,r).sol)
        end-do
        writeln("The total utilization on node ",n," is: ",
                    sum(i in SERV, q in COMP(i))CPUDEM(i,q)*sum(r in RSET(i))mapping(i,q,d,n).sol/NR(i,q,r)*repset(i,r).sol +
                    sum(i in SERV, q in COMP(i))(CPUOH(i,q))*mapping(i,q,d,n).sol)
    end-do
    writeln("The average utilization on turned on nodes = ",(sum(i in SERV,q in COMP(i),d in DOMPR,n in NODES(d))
            CPUDEM(i,q)*sum(r in RSET(i))mapping(i,q,d,n).sol/NR(i,q,r)*repset(i,r).sol +
            sum(i in SERV, q in COMP(i), d in DOMPR, n in NODES(d))(CPUOH(i,q))*mapping(i,q,d,n).sol)/
            (sum(d in DOMPR, n in NODES(d))used(d,n).sol))
end-if
!Print out information about usage costs in the public clouds
if(MODE <> 2) then
    writeln("")
    writeln("Usage costs of public clouds print out")
    forall(d in DOMPU) do
        writeln("------------- DOMAIN ",d," -------------")
        forall(i in SERV, q in COMP(i) | mappingcl(i,q,d).sol > 0.1) do
            write("Usage costs of (",i,",",q,") = ",COSTVMACT(i,q,d)*(activecl(i,q,d).sol+semiactcl(i,q,d).sol) +
                    COSTVMPAS(i,q,d)*(mappingcl(i,q,d).sol-(activecl(i,q,d).sol+semiactcl(i,q,d).sol))," ")
            if(activecl(i,q,d).sol > 0.1) then
                write("(",activecl(i,q,d).sol," active) ")
            end-if
            if(semiactcl(i,q,d).sol > 0.1) then
                write("(",semiactcl(i,q,d).sol," semiactive) ")
            end-if
            if(mappingcl(i,q,d).sol - (activecl(i,q,d).sol + semiactcl(i,q,d).sol) > 0.1) then
                write("(",mappingcl(i,q,d).sol - (activecl(i,q,d).sol + semiactcl(i,q,d).sol)," passive) ")
            end-if
```

148

```
            writeln("")
        end-do
        writeln("Total cost of usage of public cloud = ",
            sum(i in SERV, q in COMP(i))COSTVMACT(i,q,d)*(activecl(i,q,d).sol + semiactcl(i,q,d).sol) +
            sum(i in SERV, q in COMP(i))COSTVMPAS(i,q,d)*(mappingcl(i,q,d).sol-
            (activecl(i,q,d).sol+semiactcl(i,q,d).sol)))
    end-do
else
    writeln("")
    writeln("Usage costs of public clouds print out")
    forall(d in DOMPU) do
        writeln("------------ DOMAIN ",d," ------------")
        forall(i in SERV, q in COMP(i) | mappingcl(i,q,d).sol > 0.1) do
            writeln("Usage costs of (",i,",",q,") = ",COSTVMACT(i,q,d)*mappingcl(i,q,d).sol)
        end-do
        writeln("Total cost of usage of public cloud = ",
            sum(i in SERV, q in COMP(i))COSTVMACT(i,q,d)*mappingcl(i,q,d).sol)
    end-do

end-if
writeln("End running model")

end-model
```