Efficient complex service deployment in cloud infrastructure

Toan Tran Khanh

UNIVERSITE D'EVRY VAL-D'ESSONNE

Laboratoire IBISC

THÈSE

pour obtenir le grade

DOCTEUR de l'Universite d'Evry Val d'Essonne

Spécialité: Informatique

présentée et soutenue publiquement

par

Khanh-Toan TRAN

Efficient Complex Service Deployment in Cloud Infrastructure

JURY

Directeur:	Nazim AGOULMINE	Professeur, Université d'Evry Val d'Essonne, France
Rapporteur:	Pierre SENS	Professeur, Université Pierre et Marie Curie, France
	Samir TATA	Professeur, Telecom Paris Sud, France
Examinateur:	Youssef IRAQI	Professeur, Khalifa University of Science, Technology and Research (KUSTAR), UAE
	Djamel KHADRAOUI	Manageur, CRP Henri Tudor, Luxembourg
	Mario LOPEZ-RAMOS	Chef de projet, Thales Communication & Security, France
	Dominique VERCHERE	Docteur, Alcatel-Lucent Bell Labs, France

Résumé

Cloud Computing émerge comme une des innovations les plus importantes dans les années récentes. Cloud Computing est un paradigme qui transfère les données et les puissances de calcul des ordinateurs locaux en des centres de données dans le réseau. Dans la vision de Cloud Computing, les matériels et logiciels coûteux localement installés seront remplacés par des services en ligne. Ces services seront fournis aux utilisateurs à la demande et facturés selon leur usage actuel. C'est la responsabilité des fournisseurs de services de déployer les services demandés par les clients de façon automatique et efficacement. Le but du travail réalisé dans cette thèse est de fournir aux fournisseurs de services une solution qui est capable de non seulement déployer les services de façon économique, automatique, mais aussi à grande échelle.

Dans la première partie, cette thèse traite le problème de composition de service: la construction d'un nouveau service demandé par le client à partir de services disponibles dans le cloud. Pour réaliser un service complexe, le fournisseur de services doit identifier et sélectionner les composants, puis reconstruire le service demandé de manière à satisfaire les exigences en termes de qualité de service ainsi qu'en termes de coût. C'est un problème difficile (NP-hard) ; nous présentons ce problème par un modèle analytique et proposons un algorithme heuristique pour trouver la solution en temps O(V3) dans le pire cas, V étant le nombre de nœuds dans le réseau. Notre solution montre des performances améliorées de 20-30% par rapport aux autres approches.

La seconde contribution est une solution pour déployer les services dans le cloud en considérant les demandes des utilisateurs finaux (end-users). Dans un réseau de cloud, chaque service est à la destination de nombreuses enquêtes des utilisateurs partout dans le réseau, il n'est pas question que le service soit déployé sur un seul endroit. En practice, le service demandé est souvent dupliqués et distribués dans le réseau; chacun de ses réplicas servira les utilisateurs à proximité. Où placer ces réplicas et comment les coordonner deviennent important pour le fournisseur de services. Le plan d'approvisionnement selon lequel le service est dupliqué dans le réseau dépend souvent de leur demande, ce qui ne cesse pas de

changer en quantité ainsi qu'en distribution, ce qui rend le problème plus compliqué. Nous proposons une solution qui est capable de s'adapter aux changements dans le réseau, y compris ceux des demandes de ses utilisateurs. La simulation montre que face aux changements, même une petite modification dans le plan de distribution de services permet s'approcher à la meilleure solution, donc économisant largement des coûts liés au déploiement du service.

Enfin, nous proposons un système qui permet de déployer les services complexes demandés par les clients dans un cloud qui couvre différente locations. Actuellement, les fournisseurs de cloud tels qu'Amazon, Rackspace ou GoGrid offrent aux clients la possibilité de déployer leurs services dans différentes locations (régions). Faute de mécanisme de coordonner le déploiement de services, les clients doivent calculer le meilleur plan d'approvisionnement et choisir les locations manuellement, ce qui est difficile à satisfaire les besoins rapidement changées dans le cloud. Notre système est donc conçu pour automatiser cette tâche, calculant et déployant le plan d'approvisionnement optimal en respectant les contraintes du client. Ce système est très outils aux fournisseurs de services pour profiter de toutes les ressources à leur disposition, y compris les ressources extérieures fournies par les fournisseurs tiers.

Mots clés: Réseaux de nuages, cloud computing, fournisseur de services, composition de services, placement de services, optimisation, modèle analytique, algorithme heuristique, facility location.

Abstract

While appearing recently, Cloud Computing has already attracted a lot of attention from industry as well as from academic. Cloud Computing is a paradigm where the data and computing power are transferred from local computers to datacenters in the Internet. In Cloud Computing, customers can request a service and get it deployed without worrying themselves with the resources provisioning and management tasks, which fall into the responsibility of the cloud service providers. End-users can access online services from their lightweight terminals to perform their work following the pay-as-you-go principal. For that, cloud providers should be able to deploy their customers' services automatically and cost-effectively. This is a complex problem evolving different technologies such as optimization, provisioning, and security.

In the first part of our contribution, we address the complex service deployment problem based on a cloud customer's request. The objective is to allow the cloud service provider to use the already deployed services of his portfolio to build the complex service requested by a customer. Our complex service deployment mechanism takes the customer's request, identifies the necessary components, then selects and combines the service components from service provider's portfolio to construct the required complex service efficiently in term of quality of service and financial plan. This is a difficult problem (NP-hard). In this thesis, a heuristic algorithm is proposed to find a nearly optimal solution. The simulations show that our solution performs 20-30% better than other existing approaches.

The second contribution further enhances the previously proposed complex service deployment mechanism by considering the end-users' demands. In a cloud, each service may be requested by numerous end-users from all over Internet, thus it is essential that the complex service is provisioned so that it is able to serve them all. We propose a solution using duplication. In our solution, the provisioning plan consists of a set of replicas (copies) of the customer's service in the cloud, each replica serving end-users in its neighbourhood. Our solution is capable of adapt to the fluctuation in the network, including the changes of the network resources and the end-users' demands. The simulation reveals that while facing the

changes, even small modifications in the provisioning plan could result in a huge improvement, thus our proposal effectively reduces the cost associated with the service deployment.

Lastly, we introduce a system design that uses the proposed solution to deploy the customers' requests in a cloud covering various datacenters geographically distributed in the Internet. The system coordinates the datacenters of the cloud service provider and automatizes the service deployment process on these datacenters. The system is extremely useful for the cloud service provider to make use of all resources at his disposal, including the external resources that he acquires from third party cloud service providers.

Keywords: cloud computing, service composition, service placement, facility location, provisioning, analytical model, heuristic algorithm, service mapping, optimization.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor, Prof. Nazim AGOULMINE for giving me the chance to realize my Ph.D research, for his knowledge, enthusiasm and support over the years. Without his guidance, this work would not have been achievable.

I am truly grateful to all members of the thesis jury, Prof. Youssef IRAQI, Prof. Pierre SENS, Prof. Samir TATA, Mr. Djamel KHADRAOUI, Mr. Mario LOPEZ-RAMOS, and Mr. Dominique VERCHERE for their time, encouragement and constructive feedback. Their comments are very valuable for my future works.

I would also like to thank my colleagues in LRSM team, José BRINGEL FILHO, Elhadi CHERKAOUI, Pierre DELANNOY, Vamsi Krishna GONDI, Elyes LEHTIHET, Rachad MAALLAWI, Hai Dang NGUYEN, and Mehdi NAFAA for sharing a lot of ideas and for their friendly collaboration.

I am also indebted to my parents and my brother who always believe in me and encourage me to follow my dreams.

To all my friends, so many that I could not cite their names here, thank you for being with me for all these years. I greatly value their help and their friendship.

Last but not least, my deepest love goes to my wife, Viet Ha who is always by my side, encouraging and supporting me. Her presence makes Paris home.

Contents

List of Figures

List of Tables

Terminology

Application (App)

An application is software created by a customer or a service provider. In Software as a Service layer, application is also called service. Application is also built by using a VM which software is pre-installed in the image. This is a common practice to have customized images with pre-installed software stored in the network and invoke when a VM is needed.

Complex Service

A complex service is a composition of several services for the purpose of creating a new functionality. See *Service*.

Cloud customer

Cloud customer is the one who requests a complex service from a cloud provider to serve his clients (end-users). He gets payment from these end-users and pays the service provider to build his complex service. See 2.7.

Service provider

A service provider is the one who receives the requests of a cloud customer to build a complex service. In many cases, the service provider does not own the resources that are used by services; he purchases them from the infrastructure provider. See 2.7.

End-users

End-user is an user (subscriber) of a complex service, who actually interacts with the service and uses its functionality. He is a client of the cloud customer who owns the complex service and pays the latter to be able to use the service. See 2.7.

Infrastructure as a Service (Iaas)

In cloud architecture, IaaS is a layer where cloud providers provide raw resources to customers. Examples of this layer's resources include virtual machines, storages,

IP address, network connection. An infrastructure (IaaS) provider provides raw resources in the IaaS layer. See 2.7.

Instance

An instance is a copy of an application or of a part of an application. An instance can be started or terminated by the host. Each application may consist of one instances or multiple instances (distributed applications).

Platform as a Service (PaaS)

In cloud architecture, PaaS is a layer where cloud providers provide development environment to customers. Examples of this layer's resources include Java virtual machine, database. See 2.5.

Replica

A replica is a copy of a service as a whole. For instance, if a web service is composed of a HTTP instance and a database instance, then the replica of this web service is also composed of a HTTP instance and a database instance.

Service

In cloud environment, a service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistently with constraints and policies as specified by the service description*. Service is usually an application or an entity that is made accessible (to public or a selected groups of users) and to provide a particular software functionality.

Service Level Agreement (SLA)

A SLA is part of a service contract in which the level of expected service is defined, usually in measurable terms. The SLA commonly contains the criteria that the two parties have to assure, and the penalty in case that these criteria are not met.

Service portfolio (Service catalogue)

A service portfolio is a catalogue of basic services that service provider can offer to his customers. Based on this catalogue, customers may make demands for one, several or combinations of these services.

Service provider

A service provider is the one who receives the request of cloud customers to build a service. In the second part of this thesis (Contributions), service provider is con-

*OASIS SOA Reference Model TC

tracted by cloud customer to build a complex service to serve the end-users of the latter. He gets payment from the cloud customer for his services and resources that used to build the complex service. See 2.7.

Software as a Service (SaaS)

In cloud architecture, SaaS is a layer where cloud providers provide ready-to-run application to customers. Examples of this layer's resources include Facebook's apps, Google's apps. See 2.5.

Tenant

A tenant is a customer or a customer's application/system that holds the resources/services. A tenant is usually not an end-user, but a customer whose application serves multiple end-users. The term "Tenant" is usually put in the context of "Multi-tenancy" which signifies the system where the same resources are used by multiple customers - the tenants.

Virtual Machine (VM)

A virtual machine is an isolated guest operating system installation within a physical hypervisor host. Customers' application runs on a virtual machine as if running on a physical host. A VM is usually established from two parameters (or rather sets of parameters): a configured host which emulates a physical host with virtual CPU, RAM and virtual storage disks; and an image which provides a snapshot of an operating system. The image may be equipped with software to provide additional functionality.

Chapter 1

Introduction

1.1 Context and motivation

1.1.1 General context of the research

Nowadays, as companies try to cope with the rapid changes in business environment, Information Technology (IT) as a business enabler is a key for success. IT has become an essential part of every company activity regardless of their business sector, IT-related or not. It enables companies' employees to work together across geographical distances, faster business process, better coordination between activities and customers. As business grows, the companies' IT infrastructures also evolve. Although IT is supposed to play a support role, big companies see themselves in need of colossal investments into building and managing their IT infrastructure. These huge investments are poured into expanding these infrastructures to keep it up-to-date with the expanse of companies and changes of business. However, how well this funding is spent is still in question. Often IT resources are underused: HP and IBM researches estimate the usage for datacenters at about 10% to 35% of their maximum capacity, while desktop PCs usage is less than 5% of the overall capacity [15, 22].

In this context, Cloud Computing [46, 89, 104, 122, 135] comes as a new model for consuming IT resources. In this model, IT resources are shared among different tenants (different users), thus effectively increasing the resource usage efficiency. Moreover, Cloud Computing facilitates the business set-up by offering the possibility to fast and easy deployment of IT systems and breaking down the overhead capital investment. In traditional approach, companies need to estimate their usage and invest an important par of their budget into the expected IT infrastructure. However, this often results in over-estimating future

resources usage; the higher the uncertainty is, the more their estimation exceeds the reality, which leads to high investments and waste of financial capacity. Good ideas sometimes find it hard to overcome the need of huge investment to make it a reality, especially for small companies. With Cloud Computing, companies can avoid asset acquisition by purchasing resources from Cloud providers and paying only for what they actually used as they run the business and not more. Thus it transforms expensive and heavy capital expenditure (CAPEX) into more flexible operation expenditure (OPEX) and eventually reducing overhead investment [74].

Cloud Computing is a new trend that moves computing and data away from desktop and personal PCs to large datacenters. Instead of investigating in expensive computers and software, companies can dynamically subscribe for Internet-based Cloud services. Companies' data and applications are no more necessarily hosted by their own mainframes and data storage systems but in large datacenters of the cloud providers. Cloud providers manage the datacenters and assure high-quality remote access for end-users as if they were accessing these services locally. Companies' infrastructure is consequently reduced to simple terminals thanks to existing high-speed network services. Therefore, companies will not have to worry about investing in new IT products and then latter having to purchase other ones in order to keep up-to-date with the business change and software evolutions. Neither nor they need to worry about building/expanding and managing their IT infrastructure when scaling up their business. They can entrust these tasks to cloud providers and focus solely on their business activities.

1.1.2 Motivation of the thesis

In this context, one of the features that make Cloud Computing highly expected is its capability of quickly and easily deploying services instances. Services can be instantiated upon customers' demands and resources are only consumed when necessary. The resource elasticity and fine metering properties feature Cloud Computing as utility computing, the pay-as-you-go concept [32]. Providers of Cloud services will be responsible for the deployment of the entire infrastructure in order to support the customers' needs in terms of type of services (e.g. mail, storage, business applications, etc). In turn, customers do not need to worry about how the services are deployed or provisioned but only about the quality of service and cost of such services externalization.

It is necessary for cloud service providers to support the various demands of the customers in terms of access to the services but also in terms of quality of service. For that, cloud service providers need to deploy a distributed infrastructure of interconnected datacenters on

which a portfolio of services is installed and made accessible to the customers through high-speed network access. As a result, customers can make demands based on this catalogue. Cloud's elasticity characteristic features fast provisioning and automatic deployment of the services following the customers' demands.

In this thesis, we aim to provide the cloud service provider a solution to deploy the complex service specified by a customer. Let consider a scenario where a cloud customer requests a video online service from the cloud service provider in order to serve its end-users. The video online service is a complex service which is composed of three separated services in the service provider's catalogue: a storage service (to store the media content), a transcode service (to convert the media into the format compatible with the end-users' terminals) and a web service (to provide web portal to end-users). The solution proposed in this thesis aims to help service provider select these basic services, compose them into the requested video online service and deploy it in the cloud.

In order to realize the complex services requested by customers, there are two important challenges that the cloud service provider has to overcome in order to respond to the customers' needs (Fig. 1.1):

- How to satisfy customers' requests while reducing their operational cost?
- Given the multiple end-user demands for a service, how to deploy the service to efficiently serve with quality all these end-users?

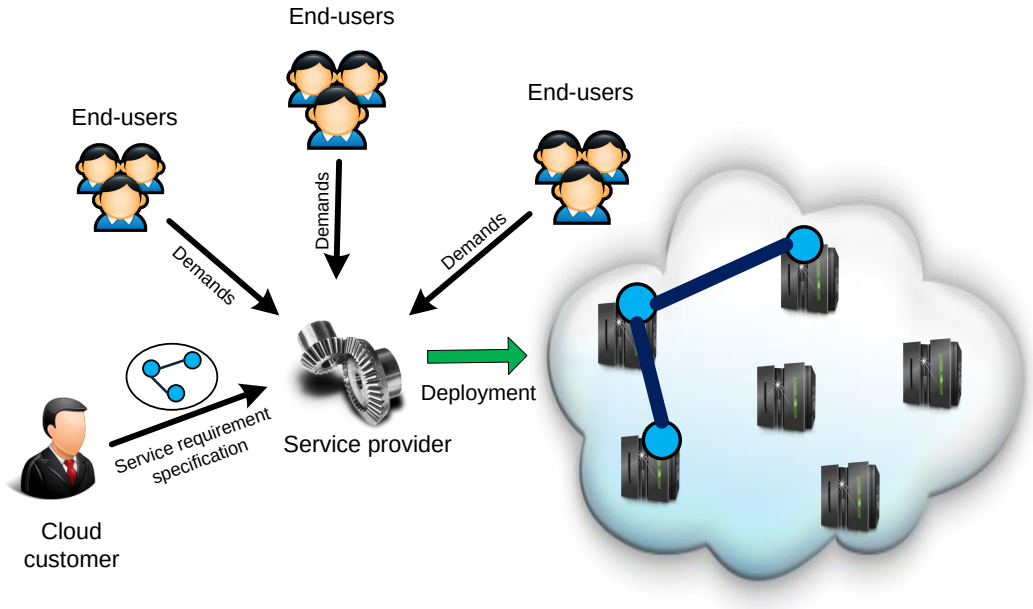


Figure 1.1: Problem illustration.

The first question addresses the provisioning mechanism to be used by the service provider

to optimally deploy the services portfolio and satisfy the customers' demands. Customers' demands can be of different types, ranging from basic standalone services to very complex composed services. Basic services can be as simple as a raw virtual machine while a complex one can be a composed set of services connected together as since in the video online example. Therefore it is important for the service provider to use an advanced provisioning mechanism capable of fulfilling the customers' demands. This is the first problem addressed in this thesis and that is, more specifically, how to optimally select basic services in a portfolio of services already deployed in a cloud that spreads over different locations to satisfy customers' demands.

This approach is in line with the current trend in the area since today big actors such as Amazon, Google, and Microsoft have already deployed several datacenters around the world that are available for their customers. Since its cloud service inauguration, Amazon already offers its customers the possibility to deploy their services in different regions. Many new actors have also emerged to share this market. Experts expect the future IT ecosystem sees clouds expand beyond the boundary of one datacenter, with many actors providing different services and qualities [40,75,112–114,125]. Thus there is a need to federate these distributed services and resources to better serve the customers.

The second problem addressed in this thesis is related to a larger view of the complex service. Since the targets of any online service are the end-users, any provisioning plan to deploy these services in the network cannot exclude them from its objectives. In this perspective, the service provider should provide his customer with an advanced service deployment mechanism that is capable to ensure a high quality of experience to end-users from various locations. To satisfy all end-users, the service provider needs to duplicate and distribute the services in different locations of the cloud system to ensure scalability. Each replica of the service may serve a group of end-users in its vicinity. This duplication of services requires additional resources and therefore increases the cost; the problem to solve for the service provider is how to design his system in terms of how many service replicas to create and where to deploy them to satisfy all end-users' demands with the required quality of service.

These two problems are the research challenges addressed in this thesis and their solutions the main achieved contributions. These contributions are briefly presented in the following section.

1.2 Contributions

With regard to the challenges identified in the previous section, the first contribution in this thesis is a new model for complex service specification and it associated deployment mechanism. Service provider can easily and cost-effectively deploy complex portfolio of services directly into the network infrastructure of the resources providers (IaaS/PaaS) and invoke them upon the customers' requests. We call this capability "service composition". With this, service provider does not need to deploy all possible services from the beginning but only a set of basic services and let customers mash-up these basic-services to create complex services as needed (similarly to web services composition). They can therefore offer wider range of services and make new business faster to set up. Moreover, this approach allows breaking down a customer's complex service request into pieces that can be provided by other (third party) cloud providers. In this thesis, we propose an analytical graph-based model to represent the complex service composition problem and an associated mechanism called Incremental Graph Mapping (IGM) to efficiently compose this complex service from already deployed basic services in the cloud.

The second contribution addresses the quality of service delivered to end-users as explained in the previous section. As Cloud Computing features high abstraction to service deployment, the actual underlying process is hidden to the users providing them with only very simple interfaces to interact with their services/resources. This abstraction allows the service provider to manage and maintain the deployed services in a transparent way to the end-users (clients of the cloud customer who requests the complex service deployment). For instance, if a physical node' load reaches certain level, the system may decide to migrate some of the supported service instances to other physical nodes that are less loaded. The whole process is executed outside the users' awareness. While users are unaware of the service's location, the latter may have high impact on the quality of service delivered to them. When the service is located closely to the end-users premise, it is easier to maintain the quality of service than for the one that is far away due to many factors, such as network instability, high delay, and high congestion possibility. Therefore, bringing the service closer to the end-users makes it more competitive, improves the quality of service as well as the satisfaction of the end-users, thus increasing the profit of the cloud customer.

The proposed approach uses duplication as a way to solve the problem. Duplication helps increase the quality of service delivered to end-users bringing service instances as close to their premises as possible. However, at the same time, this approach introduces additional costs for the service owner (cloud customer) as previously explained. The problem consists on finding the best locations for the service deployment given the end-users' locations and

demands, while trying to refrain as much as possible duplications.

Finally, the third contribution of this thesis is the Multi-site Orchestration System (MOSt). This system aims at orchestrating the deployment of complex services in a cloud infrastructure that spreads over different locations (cloud sites). MOSt implements the IGM algorithm to find an optimal provisioning plan and interacts with underlying cloud infrastructure to deploy it on-demand. MOSt preserves the independence of the underlying infrastructure, and presents a unified central point where customers can create, delete or modify their complex services. Using this system, service providers can help customers easily deploy complex services, define a provisioning plan for the service components and eventually redefine and redeploy a new plan whenever needed.

1.3 Thesis structure

The thesis is structured as shown in Fig. 1.2:

Chapter 1 introduces the context and the motivation of the research. This chapter identifies the objectives of the thesis and presents briefly the main contributions.

Chapter 2 presents a complete state of art on Cloud Computing. It presents a general view of Cloud Computing contributions from academia and industry. It enlightens what is Cloud Computing and its characteristics, what IT world is expecting from this new concept. Then, the different types of clouds as well as the main challenges are discussed.

Chapter 3 addressed the concept of complex services and service composition. Firstly, the challenges behind service composition are discussed. Then, we discuss how service composition can be applied to the cloud context. More precisely how to efficiently compose basic services already deployed by a cloud service provider to build complex services requested by customers. Therefore, the problem is formulated as a graph mapping problem, and the analytic model of the problem is delivered. Due to the NP-hardness of the problem, no polynomial solution can be found, though some particular cases can be solved in a polynomial time. Finally, we propose the Incremental Graph Mapping algorithm (IGM) to solve the problem in a close to optimal way and evaluate its performances through simulations.

Chapter 4 extends the problem of deploying services in a distributed environment to satisfy all demands initiated from different parts of the network. Instead of deploying just one complex service as in the previous chapter, a set of its replicas will be deployed to handle end-users' requests from all possible access points to the cloud infrastructure. In the context of Cloud Computing, users' demands and network environment are highly changing in such a way that current solutions prove to be inefficient to cope with these changes. Therefore,

we propose a novel solution inspired from “Facility Location Problem" (FLP) resolution approaches to find the optimal provisioning plan to deploy the set of service replicas in the network. Our solution relies on the prediction of end-users’ demands and the operational objectives of the service provider. Various simulations have been performed to highlight the performance of our solution against other contributions.

Chapter 5 is dedicated to the implementation of the Multi-site Orchestration System (MOSt). The purpose of the system is to provide an orchestration mechanism to deploy complex services over different underlying cloud infrastructures (sites). The system implements the IGM algorithm (Chapter 3) to calculate an optimal provisioning plan given the constraints and deploy it upon various cloud sites. MOSt is designed in such a way that it can interact with any underlying infrastructure cloud provider, so that the service provider can make use of additional resources to satisfy specific end-users demands that cannot be satisfied otherwise.

Finally, Chapter 6 concludes this thesis. It synthesizes the overall contributions and highlights some perspectives for this research.

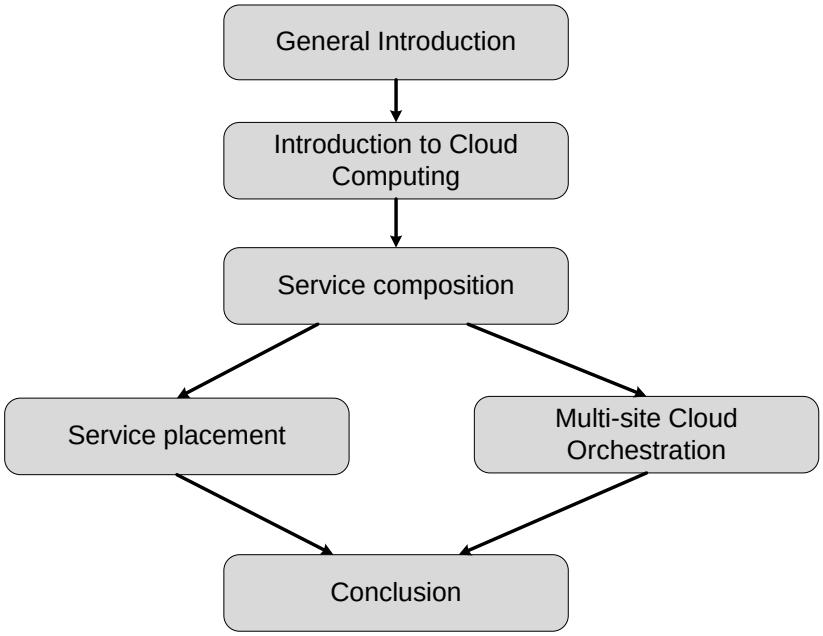


Figure 1.2: Thesis structure

Part I

Introduction to Cloud Computing

What is Cloud Computing? When did it appear? What is new in Cloud Computing? And what impact does it have to the IT world? Although making it to the market for quite some time, Cloud Computing is still a source of confusion to many. Many of us already make use of Cloud Computing that we are not even aware of. Even finding a definition of Cloud Computing that is accepted by all is difficult. This part tries to formulate and elaborate a unified definition and objectives of Cloud Computing and give a global view of what Cloud Computing really brings. Different aspects of Cloud Computing as well as the main challenges will also be presented in this contribution.

Chapter 2

Cloud Computing: Introduction, definition, models and research challenges

2.1 What is cloud computing?

Cloud Computing is a new paradigm that moves storage and computing power from desktop computers to online datacenters. It is a multidisciplinary field that includes virtualization, grid computing, power efficiency, distributed computing, service-oriented architecture. Cloud Computing promotes the use of online services and resources in a pay-as-you-go charging basis to handle IT-related tasks.

More precisely, Cloud Computing means that customers can use computing resources outside their premises and pay for what they actually use, with the possibility to dynamically provisioning new resources. It is a loose concept that covers many use cases that people are already familiar with. For instance, webmail services such as Google's Gmail, Yahoo Mail, and Microsoft's Hotmail are cloud-typed services since customers store their data (mails) on the providers' datacenters. Facebook is also a cloud-typed service: customers store their data on Facebook's datacenter and use applications from tier providers integrated into Facebook platform. Cloud Computing also offers servers and Internet access to customers to build their own applications. These services are already offered by current datacenters and grid computing. So what is new in Cloud Computing that levels it above all else that has been already done (say, grid computing)? For example, Saleforce.com that was created well before the word "Cloud Computing", is it also a model of cloud services? Can a datacenter that also offers computing and storage services be categorized a cloud computing

service? These questions are often the source of frustration and confusion about Cloud Computing [72].

Many have tried to define the Cloud Computing and its goal [26, 57, 58, 74, 124], each covers some aspects that they think Cloud Computing should be. The following definition, introduced by the US National Institute of Standards and Technology (NIST) [89], is so far the most widely accepted one that covers large aspects of Cloud Computing:

> *"Cloud Computing 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. This cloud model promotes availability and is composed of five essential characteristics (on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service), three service models (Software as a Service - SaaS, Platform as a Service - PaaS, Infrastructure as a Service - IaaS) and four deployment models (private cloud, community cloud, public cloud and hybrid cloud)."*

However, NIST also stated that as Cloud Computing is an evolving paradigm, this definition is still subject to be redefined following debates in public and private sectors [89]. Nevertheless, this definition emphasizes a clear advantage of cloud against current datacenters. Cloud Computing aims to allow quick allocation/release of resources to fit customers' needs without preliminary contract. Current datacenters require hours, or days of preparation before the resources can be allocated following customers' command. Resource scaling even needs more time and effort to seamlessly integrate into the existing system. Thus companies need to provision far ahead of their project when they want to allocate/purchase resources. If the cloud succeeds in allowing resource scaling almost instantly without human intervention, this will help companies to start their project with small amount of resources, then scale up or down the resources to meet the business trend.

It is hard to trace back when the term "Cloud Computing" was first used. To the best of our knowledge, the first document ever stating "Cloud Computing" as it is used today is a lecture titled "Intermediaries in Cloud-Computing" by Chellappa at INFORMS meeting 1997. Then latter the term is restated in his article published in the International Journal of Network Management in 2002. He describes Cloud Computing as "a dynamic computing paradigm where the boundaries of computing are determined by rationale provided by technological, economic, organizational and security requirements" [33]. The term "Cloud

Computing" only became popular in public around mid-2007 after Amazon had opened its services Amazon Elastic Compute Cloud EC2 (Fig. 2.1).

Figure 2.1: Google search statistics of ’Cloud Computing’ - Source: Google

2.2 Why “Cloud"?

The word “Cloud" represents the abstraction that hides the complex underlying processes in the infrastructure. Many believe that the term is employed because a cloud icon is usually used as a metaphor of Internet and Telecommunication networks in diagrams. Since the complex Internet/Telecom infrastructure is not easy to capture, the cloud icon is widely used to abstract the most important characteristic of Internet: the connection between any end-points in the network. With a similar purpose, Cloud Computing tries to provide users with simple abstraction that hides the actual processes that run in the underlying infrastructure.

The cloud abstraction allows users to send commands and interact with the system through a thin terminal, like a web browser, although the actual process may require more than one server or service running at the same time. In this respect, Cloud Computing is similar to Grid Computing, which also provides a single interface to users while splitting the task into multiple threads and processes (each thread running on a server of the grid). However, they are different in nature. Grid Computing is destined to processing tasks by aggregating the computing power of distributed servers in the grid. Grid Computing aims to provide standard set of services and software that enable the collaborative sharing of federated and geographically distributed compute and storage resources [50]. Current cloud provides resources from massively concentrated datacenters. Cloud Computing is utility-driven: resources are purchased if needed and can be released whenever the customers are done, with

the capability of rapid scaling up and down. Even though Grid provides middleware interface, it typically acts on top of actual hardware, and rarely rely on virtualization, thus the sole purpose is to provide computing resources as a whole for its community. Cloud Computing, however, is based on virtualization to isolate customers' resources, and oriented towards the general usage: we can even build a grid inside a cloud. Cloud Computing not only aims to provide low level utility such as computing, storage or bandwidth resources in infrastructure level, but also higher level solutions such as stacks in platform or applications in application level.

2.3 Cloud characteristics

Researchers have identified many characteristics of Cloud Computing. In our vision, there are five characteristics which particularly distinguish Cloud Computing from traditional datacenters: on-demand service, multi-tenancy, elasticity, high abstraction level and fine metering.

- On-demand service: cloud customer can unilaterally provision resource capacity as needed automatically without the administrator's intervention. In cloud computing, customers do not have any control over the underlying infrastructure however they are provided with a configuration interface that allows them to adjust their resources on-demand (this is automatically enforced after by the underlying cloud management mechanisms).
- Multi-tenancy: in the cloud, a customer who consumes a service is called a tenant of the service (resources, application, data, etc.). In single-tenant model, each customer (or an application) has a set of dedicated resources to serve his needs. To reduce the cost, cloud providers use the same resources to serve multiple customers (even simultaneously): this feature is called multi-tenancy. This applies not only to resources, but potentially also to data and services. However, each customer has a unique profile, and to avoid security threats, cloud customers do not share the same environment. They are served within isolated environments thanks to virtualization.
- Elasticity: elasticity is the capability of the system to quickly scaling up or down resources allocated to customers. Elasticity features the quick reaction of cloud resources to the customers' demand. Once the demand is received from the customers, the cloud infrastructure should take only a few minutes to fulfil it. There are two types of elasticity: horizontal elasticity and vertical elasticity. Horizontal elasticity is the capability of scaling resources by adding/ releasing instances/virtual machines. Vertical elasticity is the capability of scaling resources by adjusting the size of the

running instances/virtual machines.

- High abstraction level: cloud users are provided with simple interfaces that hide most of the technical details of the underlying network. It allows customers to use the cloud services without any technical knowledge of the underlying network resources management. For instance, customers can store their data in a cloud and retrieve it using simple commands without knowing how the data is stored, duplicated and protected. These are under the responsibility of the cloud provider.
- Fine metering: the cloud allows users to monitor and pay the resources/services they use in a fine grain (*pay-as-you-go* basis). Contrary to traditional datacenters which customers have to pay on a monthly or yearly basis, cloud providers offer payment per hours, which allows customers to manage their budget according to actual used resources, not on long term provisioning. Unlike software delivery model where deployed application is outside the scope of a service provider, a multi-tenant application is fully under his control. Therefore the cloud provider can gather the behaviours and operational information of large community of users, thus making the service management and improvement easier and better to adapt to the customers' need.

Cloud Computing offers the possibility of scaling up or down the resources in brief delay. The high abstraction level allows companies to adjust their resources from a simple interface without being involved in the burden of reconfiguring their system. Cloud service providers are responsible for managing the underlying infrastructures. Therefore adjusting the resources becomes a smooth and simple task for companies. The latter can automatically scale up or down their consumed resources following their actual usage, contrary to traditional datacenter services which are based on long term provisioning.

In the traditional resource provisioning approach, companies have to well study the clients' demands in order to dimension their hardware or leased resources from service providers. Indeed, there are two main strategies for resource provisioning. The first one is over-provisioning and it is the approach often used by companies. Over-provisioning strategy aims to allocate resources so that they can serve even the peak load (i.e. when number of clients is maximal). The purpose of this strategy is to ensure a high quality of service to the customers. However, as clients' usage is variable and may change rapidly due to some special events or time tables (cycle day/night, seasons, etc.), this results in underused allocated resources. For instance, traffic on Nasdaq.com follows the week cycle, with highest workload in the middle of the weeks, and lowest one on weekends (Fig. 2.2). Hence, over-provisioning resources of Nasdaq results in waste of resources in weekends.

Another strategy is under-provisioning, which allocates resources at a level that is below the peak load. This strategy aims to reduce the allocated resources while at the same time

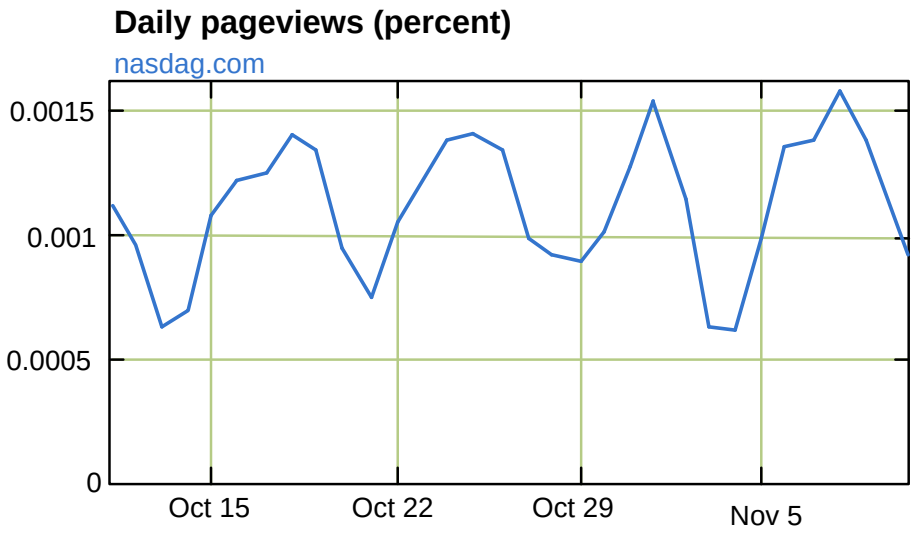


Figure 2.2: A typical workload evolution of Nasdaq.com. The result is observed in one month period. Source: Alexa.

increasing the risk of violating their level of quality of service provided to clients during peak periods. Although this strategy is more economical to companies, the risk is the departure of clients due to unsatisfactory quality of experience. While over-provisioning cost is easy to measure, under-provisioning cost is more complicated and more serious.

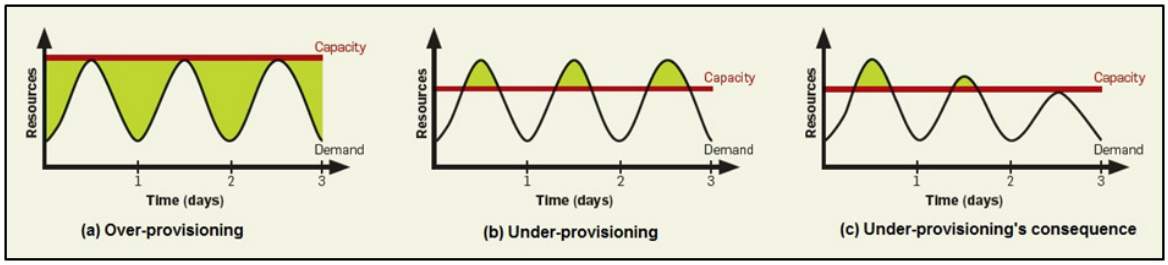


Figure 2.3: Over-provisioning and under-provisioning in traditional datacenters. (a) Over-provisioning: resource capacity is able to capture the peak load, but most of the time the resources are underused. (b) Under-provisioning: the capacity is below the peak load. (c) Under-provisioning's consequence: the clients leave the company due to unsatisfactory service. This will continue until the number of clients is low enough so that the peak load is below the capacity threshold. Source: [18]

Fig. 2.3 illustrates an example of the risk of under-provisioning: the clients leave the service due to unsatisfactory experience during peak periods, until the workload is below purchased resource capacity. Due to this risk, companies prefer using over-provisioning despite the waste of resources. Real world estimation of average server utilisation in datacenters is about 10% to 35% [15]. One of the reasons is that the peak load is a lot higher

than average load, thus most of the time the resources is underused. Another reason is that they have to provision for future potential subscribers.

The elasticity of cloud computing allows companies to quickly sale the allocated resources in order to meet the services' demand in brief moment. Thus companies don't need to provision far ahead and can quickly allocate/release resources once the services' demand changes. This removes the up-front resource investment and scale resources close to the actual usage. Companies pay for their consumed services/resources in short term usage, thus removing the long-term commitment. The fine metering and pay-as-you-go billing policy help companies adapt the resources budget to their actual usage, effectively erasing the downside of over-provisioning.

Another characteristic of cloud computing is the "appearance of unlimited (resource) capacity" [17], or known as "illusion of infinite resource available on demand" [11, 126]. Under traditional approach, once the company makes a contract with a service provider, the resources are allocated to the company and it is their responsibility to well manage them. Allocating more resources for the services is a complex process, which implicates many activities to recalculate the budget dedicated to expand system, provision resources, and integrate new resources into the current system. This process takes time and effort, and should only be used in the case of business objective modification (e.g. the company decides to expand their services following its success or reduce the portfolio of services due to lack of clients). Therefore, designing or modifying system is always done in restricted resource constraints. In Cloud Computing, as companies can allocate/ release resources at will, the notion of 'capacity' is literally ignored. Indeed, cloud providers invest into massive dantacenters so that they can satisfy any resource demand of companies. Furthermore, scaling resources would cause no problem of resource integration thanks to virtualization technology. Thus companies can always expand their system as if pooling from an unlimited source.

A recent vivid example of the success of Cloud Computing is Animoto which hosts its service on Amazon's AWS cloud. When it opened its service to Facebook users, nearly 750,000 people signed up in roughly 3 days, which resulted in a spectacular growth to 3,500 Amazon EC2 servers from 50 servers previously used. At the peak, almost 25,000 people accessed the services in a single hour [53]. This growth surge was unpredictable and well beyond the traditional provisioning mechanisms' ability. It would have been impossible for the company to anticipate such a demand without the flexibility in resource scaling of Cloud Computing. Finally, Animoto only paid about 10 cents per server per hours and some marginal expenses for bandwidth and related services during the surge.

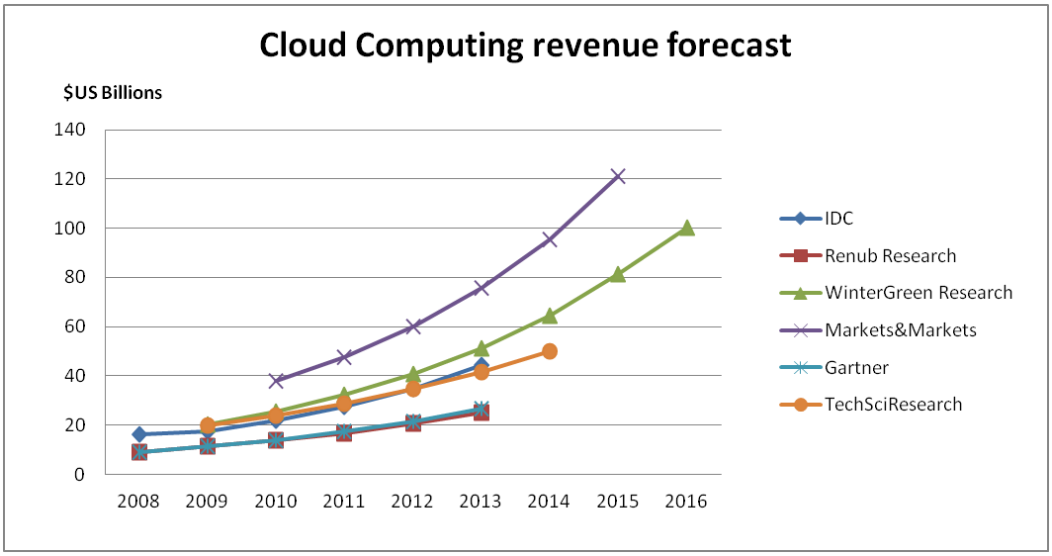


Figure 2.4: Cloud Computing revenue forecast according to market intelligence. Sources: [28, 85, 110, 127, 133]

2.4 Model for future IT?

Despite being a new concept, Cloud Computing has already gained significant attention from IT industry which expects it to play a great role in the future market. Due to the disagreement of categorizing which services are cloud-typed and which are not, the market intelligence firms made different reports on Cloud Computing-related expense and prediction of recent future (Fig. 2.4). According to IDC, Cloud Computing revenue is $16.2B in 2008, $17.4B in 2009 and expected to achieve $44.2B in 2013. This is equivalent to 26% of growth annually (Compound Annual Growth Rate - CAGR). Gartner evaluates the Cloud total revenue in 2008 to be $9.1B and predicts the revenue of $26.6B in 2013 (CAGR 24%). Using the data provided by Gartner, Renub Research evaluates Cloud revenue is $9.10B in 2008, $11.28B in 2009, and will reach $25B in 2013 (CAGR 22%). Winter Green Research predicts a growth from $20.3B in 2009 to $100.4B in 2016 (CAGR 25.6%). Markets And Markets expects Cloud market will raise from 37.8B in 2010 to $121.1B in 2015 (CAGR 26.2%). Despite of the difference in estimation values, all are expecting a high growth rate (CAGR between 22% and 26%) - six times of IT growth rate in the same period, which promises a good future ahead for Cloud Computing.

So what is the reason that may support the steady growth of Cloud Computing and will it become the model of future IT? As previously discussed, customers would benefit greatly from the elasticity capability and pay-as-you-go billing basis. Even if the cost rate for cloud services is higher than the same services in tradition IT, it makes up by the flexibility of

service usage. Compared to over-provisioning approach which provisions resources for peak load, Cloud Computing allows dynamic resources consumption that comes close to actual usage, thus greatly reducing the resource volume-time, and consequently the usage cost. Moreover, as resource scaling is done automatically in the infrastructure, adjusting resources to the customers' needs is quick and easy, further eliminating operation time. For instance, when the Cloud Power web observed a sudden growth in traffic due to an ads campaign which saw the traffic up to 700% normal rate during three days of the campaign, the engineering team decided to double the servers used to handle the traffic. They didn't have to allocate VMs, install and patch OS, or install applications, but only to change one value in a XML file and the cloud platform took care of the rest. The whole campaign costs only 70 USD and about 5 minutes of operation time [111]; the solution was a lot cheaper than buying high capacity server and managing it.

Not only companies which business relies on their online services would benefit from Cloud Computing. As general-purpose, any company that benefits from IT technology will also benefit from Cloud Computing. Companies will have their internal system hosted on the cloud, and only need thin devices for their employees to access applications that are usually installed on PCs. While hardware price is continuing to decrease, it is not the case for software and integrated systems. Many companies still use old software to do their work because it is expensive to buy a new software or difficult to migrate the data into the new software. With Cloud Computing, they can literally ignore the capital expenditure on hardware, software and maintaining cost, which falls now into the responsibility of cloud service providers, and choose the solution that they desire. Suppose that a company invests $1,000,000 into the mainframes for building their own system. This amount must be paid before the system is actually deployed and used and the company expects that the mainframe will hold their performance for 10 years. The situation can be worse since no one can be so sure that the mainframe can be as efficient after 10 years in use, or the company will not have to purchase new mainframes of better solution to remain competitive. With Cloud Computing, the company does not pay all the money at once, but disbursed along 10 years, thus greatly improving the return on investment (ROI). Moreover, they can benefit from the solution upgrade of cloud providers and moving to another solution would not raise extra charge. Thus even if the price per unit in cloud is higher than purchasing resources, using cloud service is still safer and less costly than purchasing and operating/maintaining the system, particularly small and medium companies who could not have the IT expertise. In a way, Cloud Computing transfers the risk of owning the system from companies to cloud service providers.

For cloud service providers, gain of customers means their opportunity. By investing cap-

itals into mass datacenters at low-cost locations, and having a single solution licence for all their customers, cloud provider may reduce the cost of deploying systems by the factor of 5 to 7 in term of hardware, software, power consumption, bandwidth, and operation/maintain lower than if separately deployed by customers alone [17, 18]. These factors are added by resource multiplexing that will increase the resource efficiency manifold.

Aside from Software as a Service (SaaS), in the present Infrastructure as a Service (IaaS) and Platform as a Service(PaaS) offers are mostly addressed to small and medium companies. This is understandable since these companies have limited budget for their IT infrastructure, and big companies have their own infrastructures that are not easily discarded. However, Cloud providers and market intelligence firms are expected that in 5-6 years. Although there are many obstacles for Cloud Computing's prosperity such as security, business continuity and service availability, Cloud Computing is hopefully expected to be the future of IT.

Cloud sympathizers often refer to the electrification process in US at the transition of the 19th century to 20th century to support this belief. Like IT, electricity is general-purpose technology that assists in all sorts of companies whatever their activities. Since the early stage, companies were well aware of the necessity of electricity to their business. To have enough power to their use, big companies had to contract suppliers such as General Electric and Westinghouse and hire electric engineers to build their own generator and integrate it into their system. By 1907, 60% electricity used in US were generated by these factories [29].

Small and medium enterprises (SME) that could not afford building a generator have no choice but to buy electricity from nearby suppliers with high price. These increasing demands, along with technology factors, led to the creation of centralized electricity stations and large electricity utility grid. This, in turn, lowered the electricity utility price to the point that even large companies found it irresistible to ignore. Although it was not easy to abandon their investment into generators, large companies eventually connected to the electricity grid.

Companies in early era of electrification and IT era have a lot in common. They have to rely on technologies to be competitive and increase efficiency. In electrification era, companies had to purchased pieces of electric components, figured how to integrate them into their factories and struggled with their old equipments which were incompatible with new systems. Companies in IT era are worried with purchasing IT products, using them in their systems and solving problems with legacy products. Billions of dollars are invested into support and maintenance functions [62]. The Cloud Computing would be the solution for these problems as the centralized generator stations were for the electricity problem a

century ago.

In the end, electricity centralized station and Cloud Computing are rational consequences of the larger processes that are essential to the industry: the specialization and externalization. By investing massive capital into a specialized business, companies can provide the product with less cost and higher efficiency. Other companies then give contracts to these companies to provide them with the necessary products that will be integrated into the final business chain. This process is called externalization or outsourcing, a common practice that liberates companies from having to handling every details of their business chain so that they can focus on their main business goal or simply helps them overcome obstacles that they are unable to do by themselves.

2.5 Cloud architecture

Initiatives on cloud have reached an agreement on the high level design of a cloud system. The Cloud follows the layered architecture (Fig. 2.5). The top layer is the application layer directly accessed by end-users. The middle level is the platform level where development platform (operating system, database, language tools, etc.) are supplied for companies to build and run their services. At the low level is the infrastructure level where cloud providers offer various computing resources (e.g. servers, storage, routers) which are virtualized environment on top of their hardware. One example of this architecture is the applications of Facebook developed on Heroku which hosts their platform on Amazon EC2 servers [25]. Fig. 2.6 illustrates an example of implementation of cloud services from infrastructure layer to application layer. Details of each layer are presented in the following paragraphs.

Infrastructure level: Infrastructure as a Service (IaaS)

At the bottom level, IaaS providers provides raw resources to customers such as CPUs, storage, servers, switches, routers. IaaS providers offer resources in form of virtualized environment on top of their actual hardware. As the technique of generating alternating current changes the electrification process, the virtualization is the key technology for to Cloud Computing. In virtualized environment, each customer's resources are separated from others', allowing cloud providers to allocate or shut down, scale horizontally and vertically resources as requested without affecting others'. It also enables IaaS providers to migrate (transfer) customers' resources and integrate new hardware if needed without customers' notice.

At this level, customers only have control over the resources they purchased (e.g. they can install OS and various components), but have no control over the hardware below other

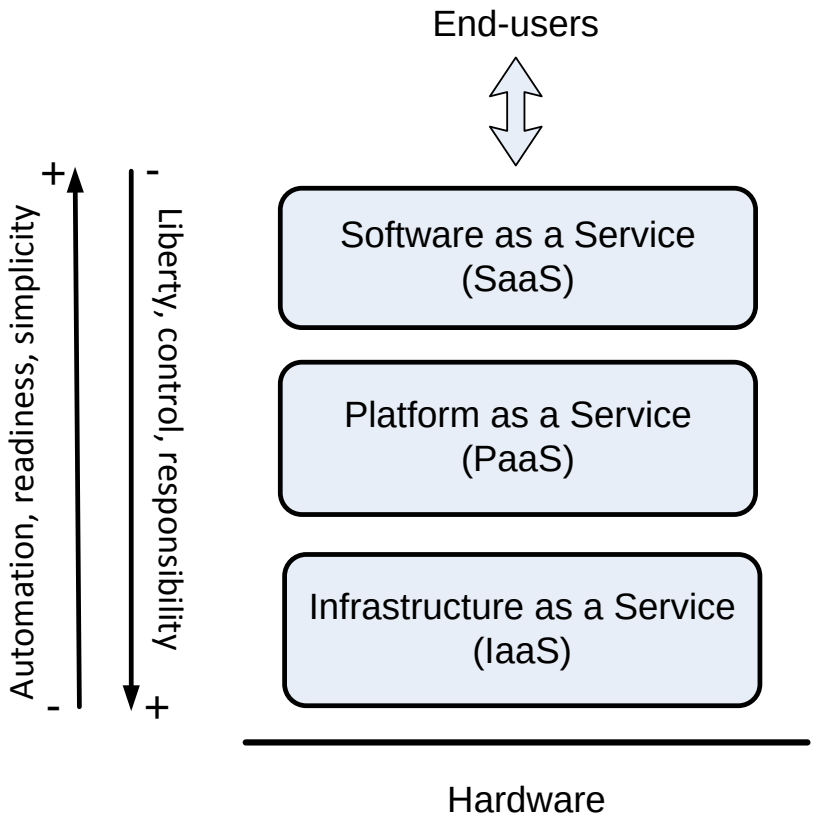


Figure 2.5: Cloud layer architecture

than adjusting (virtual) resources using the interface provided by cloud providers, e.g. determining exact location of their resources in the network. This level offers a great freedom to customers to deploy their infrastructure and build their services by any mean they want (see Platform level below). However, it transfers more responsibility to customers to manage their resources and services.

Examples of IaaS offering include Amazon EC2 (computing), Amazon S3 (storage), IBM Smartcloud, Eucalytus and Rackspace Cloud.

Platform level: Platform as a Service (PaaS)

PaaS providers provide customers with a platform comprising the necessary components so that customers can build and host their applications. PaaS providers supply customers with API that comes with their platform to build their applications. The API features out the key capabilities of the platform offered by the PaaS providers. Customers can control the behaviour of the hosting engine which executes the applications through the API but have no choice over OS and supported software components (e.g. database, framework, middleware) of the platform.

Customers who just want to build applications and services without being concerned with the infrastructure would choose PaaS instead of IaaS. IaaS offers customers with more control but also requires more responsibility and effort to deploy and manage the environment. PaaS providers restricts the control of their customers over the environment they provide but in turns, they resume the responsibility of managing the platform to run applications efficiently, from scaling resources to compiling and running applications. It's like providing customers with Lego pieces ready to be joined together as they like.

At current state, these provided APIs are usually specific to the PaaS, and applications build on the platform of one provider can hardly be used on others. For instance, Google App Engine is dedicated to Web services only, and Microsoft Azure only executes applications built in .NET framework or PHP (although Java and Ruby developer now can use special SDKs to integrate their applications with Azure AppFabric).

Examples of PaaS offering include Google AppEngine, Microsoft Azure, Heroku, Sun Cloud, Force.com, and Rackspace Cloud.

Application level: Software as a Service (SaaS)

The highest level of cloud architecture is the application level. These applications are complete, developed on top of cloud infrastructure to serve users in their business goal, not to provide cloud features. At this level the customers are end users, who generally do not need high IT knowledge to work with applications. Users can interact with the application from thin interface, such as Internet browser.

Examples of SaaS offering include Salesforce.com, Google App, and Facebook.

The lower the layer is, the more responsibility is transferred from cloud providers to customers, but the more liberty and control the customers receive. At the lowest level (IaaS), cloud providers only supply customers with basic resource metric monitoring such as CPU/Memory load, access, link load. It is the responsible of customers to control their systems, elasticity feature remains as simple as creating/removing VMs. They have to well design their systems to make use of elasticity, implementing load balancing policy and control their components. At the PaaS layer, cloud providers may provide application-centric monitoring, policy-based load balancing, automatic scaling and firewall. At the top layer, customers only use the applications, and all deployment and management tasks fall into cloud providers' responsibility. At the other side, the level of simplicity and readiness is lowered by layer. IaaS customers have control over their systems even in operating system layer. They can deploy their systems for general purpose, not limited as in PaaS.

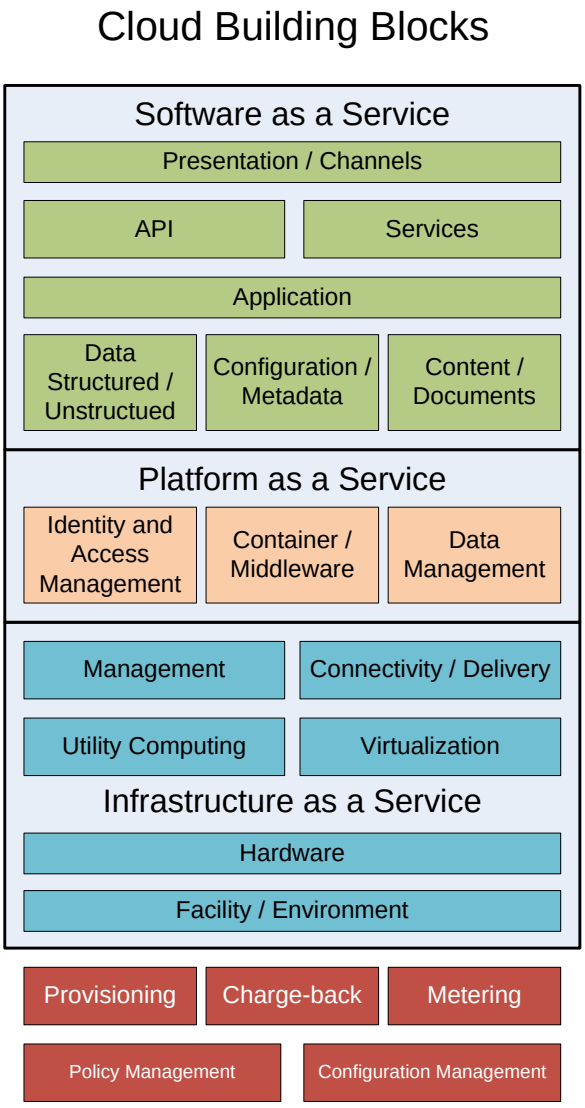


Figure 2.6: Cloud implementation example - Source: Virtualgalaxy [55]

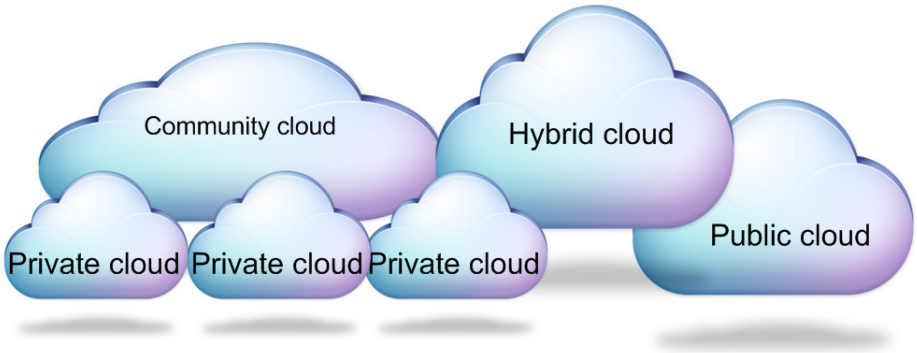


Figure 2.7: Types of clouds

2.6 Type of clouds

There are several types of clouds that could be deployed, depending on the number of customers they serve and the business model. It ranges from private clouds to public clouds, with the mix of community clouds and hybrid clouds. In the early stage, Amazon built an IT system solely destined for their retail activities; this early form of system is considered as a private cloud, which is owned by Amazon with open interfaces to its customers (and retailers) only. With the maturity of the technology, Amazon became confident to offer its IT expertise to customers by opening its infrastructure; its system became a public cloud [121]. This evolution is different from services such as Salesforce.com, Facebook or Google Apps where their owners' purpose is online services (SaaS) to public from beginning.

Public cloud

As the name suggests, the public cloud provider offers their cloud features to public. Business customers can then purchase and use these features for their business. Public cloud is what we truly expect a cloud to be with all its features. Public cloud is multi-tenant nature, meaning that the customers share the same resources. As mentioned above, the virtualization technology allows isolating customers, thus the provider can serve a customer's request without affecting the others. Examples: Amazon EC2, Microsoft Azure, Google AppEngine, IBM Smartcloud.

Private cloud

In contrary of public cloud, private cloud is owned by one organization and its cloud resources are not open to other organizations. However, private cloud owner may give their clients restricted access to the cloud. Private cloud may be operated by the owner or by a third party. GoGrid [60] provides the private cloud offer which is built upon their dedi-

cated physical hardware . However, GoGrid charges dedicate servers by specific server in monthly or yearly basis like in traditional hosting service instead of pay-as-you-go basis. Quickly scaling resources above the capacity of the rented dedicated servers is impossible (apparently 2 business days are needed for a new dedicated server to be deployed). There are many debates over private cloud, whether these so-called "private clouds" are truly clouds.

Firstly, private cloud may refer to private platform of services that the company offers to its customers. In this case the private cloud can be seen as SaaS platform where customers only use services for their business, with no control over the computing resources of the cloud.

Secondly, private cloud may refer to resources pool that company build for their own use. Here again, these are many debate whether it is worthy categorized as "cloud". The UC Berkeley RAD Lab [18] does not qualify small and medium datacenters, which they call "private datacenters", as "cloud". They argue that the only extremely large datacenter can benefit from the economy of scale (massive quantity lowers price per resource unit). Small and medium sized private datacenters still have the shortcoming that Cloud Computing is designed to overcome: high up-front investment, ineffective operating/maintaining cost per resource unit (including maintaining IT expert staff on premise), over-provisioning of resources. These datacenters could neither provide high scalability or high resource usage efficiency from multiplexing as large datacenters running thousands of services would.

In our vision, we consider these datacenters as cloud. Although they may not have as many benefits as the public cloud or large private cloud datacenters do, they are still built following the cloud architecture principles, including the elasticity and high abstraction features. Building private cloud means that the company's infrastructure is flexible so that they can adjust the resources between services, or extend/replace the resources at ease. New services can be deployed, and inefficient services can be removed from the system without much human effort.

More importantly, building a cloud-like infrastructure allows companies to seamlessly integrate into the public clouds if they need more resources (see Hybrid clouds below). In many cases it is important for companies to host their confidential data on their own datacenters for security purpose - which is considered as a weakness part of multi-tenancy model of public cloud [69]. The hybrid cloud (see below) allows them to help keep their data safe while still have the benefits from using the resources from public clouds. In the migration process from traditional corporate network to cloud computing, companies will not be willing to abandon their systems easily. It is natural and economical for them to first build up a cloud-like architecture inside their private infrastructure then progressively migrate to public offer [102]. Thus private cloud is preferred by many at this stage of Cloud

Computing evolution, which leads to many private cloud offers. For instance, Ubuntu OS from 8.10 version is shipped with Eucalyptus software that allows users to build their own private cloud. They can also extend their cloud to Ubuntu One or Amazon EC2 [61].

Community cloud

Community cloud refers to the type of clouds that are owned by several private contributors. These contributors form a community with common purpose and deploy a common cloud infrastructure. In this aspect, it resembles private cloud in the sense that it destined to very limited actors and is not open to public. Therefore community cloud suffers doubts and confusions whether we should regroup it into the same category with private cloud. In community cloud, however, the contributors are independent but share common infrastructure. They can be filial companies of the same group, group of universities or laboratories, etc. Unlike public cloud where the offered resources are the propriety of the service provider alone, the community contributors offer their physical resources to build a common cloud on which they can deploy their services and collaborate together. Community cloud is considered as a form of grid community in the cloud.

Hybrid cloud

Hybrid cloud refers to the composition of private cloud (or community cloud) with public cloud. While public clouds offer many benefits to customers, they also have some issues that remain challenging, such as security and latency. Since customers generally are not aware of underlying infrastructure, they have no idea where their data are located and how they are protected. Thus it may be subject to attacks from outsiders, or even from other customers in the same cloud, or legislative threat. Service latency is another issue. Although the data storage is increased exponentially in recent years, the network bandwidth does not observe such improvement. Therefore data transfer over the network becomes an important issue for all IT services, not only to Cloud Computing. For a model totally based on Internet such as cloud, this is a major drawback. The UC Berkeley researchers calculated that if they wanted to send 10TB data to Amazon datacenter in Seattle, it would take more than 45 days for the transaction to finish [18], whilst shipping 10 1TB disks would take roughly one day or two!

Extending the private cloud to public cloud in a hybrid mode while retaining what is necessary inside the company premise is an effective solution to benefit from the public cloud while overcoming its drawbacks.

Yet, hybrid means compatibility and interoperability, and that public cloud providers should offer APIs to interoperate with other cloud solutions. However, in the lack of cloud standardization in computing resource control API, IaaS and PaaS hybrid clouds are difficult to achieve. The problem is often mentioned as "Vendor-locked in", which refers to

the fact that customers have to build their systems upon the APIs provided by one cloud provider, and they can hardly be used with another cloud. There are currently some efforts to standardize resource description for Cloud Computing such as OVF [44] or OCCI [100]. Some software is also proposed to build private clouds that are compatible with major public clouds. Eucalyptus [93, 98, 99] is an open source software compatible with Amazon EC2; private cloud built on Eucalyptus may extend into Amazon EC2 cloud. Likewise, OpenStack [6] is an open suite to build a cloud which is compatible with Rackspace [7] and Amazon EC2 public cloud.

2.7 Cloud Business Model

In the cloud business model, we consider that each actor has a role associated with a service it provides in one of the cloud architecture layers. Of course one company can offer different services in several layers, such as IBM with their SmartCloud service suite ranging from IaaS to SaaS services.

The US National Institute of Standards and Technology (NIST) defines a cloud customer as “an actor that maintains a business relationship with, and uses services from a cloud provider". The cloud customer “browses service catalogue from a cloud provider, requests the appropriate service, sets up service contracts with the cloud provider, and uses the service" [70]. The services may vary from resources in infrastructure level (IaaS), platform in platform level (PaaS) or applications in application level (SaaS).

Also defined by NIST, a cloud provider, also known as “cloud service provider", is “a person, an organization, or an entity responsible for making a service available to cloud consumers". A cloud provider may be very well a customer of another cloud provider of the same layer or from another layer below. For instance, a SaaS provider may be customer of a PaaS provider who hosts his platform on the infrastructure provided by an IaaS provider. The latter may not possess physical hardware but allocate resources from other IaaS providers and build a homogeneous infrastructure on top of these IaaS providers’ infrastructures (he is sometimes called cloud reseller). Therefore from end-users to IaaS providers who own physical hardware there is a chain of customer - provider relation.

Definitions used in this thesis

To clarify the role of these actors in this thesis, we introduce some definitions of these terms that the remaining of the thesis will refer to (Fig. 2.8).

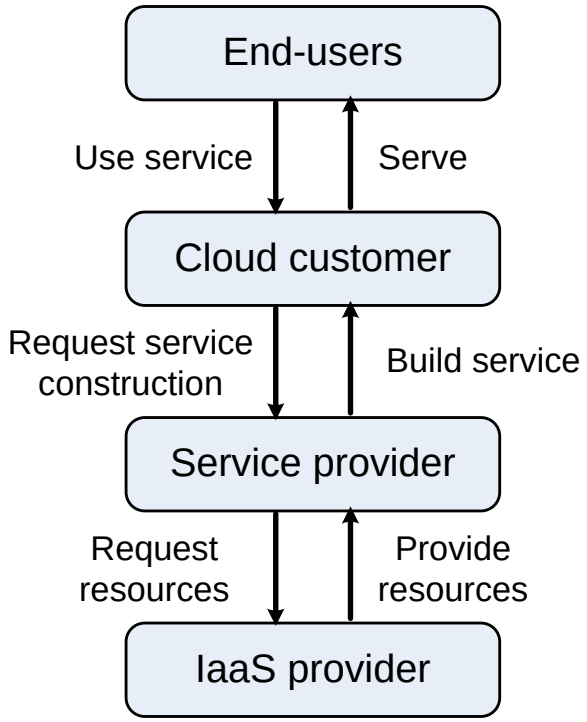


Figure 2.8: Roles of different actors in this thesis

- *Cloud customer*: Cloud customer is the one that requests service deployment from a service provider to satisfy his clients (end-users). He is the owner of the deployed services. He charges his clients to use his services, and pays the service provider for building, deploying and maintaining the services.
- *End-user*: End-user is an user of a (complex) service, who actually interacts with the service and uses its functionality. End-user pays cloud customer who owns the service.
- *Service provider*: Service provider takes the contract from the cloud customer to build a service following the latter's needs. It offers its customers to deploy the latter's services in his service infrastructure. There are often cases that service provider does not own the underlying infrastructure from which he draws resources to build the complex service as requested by the cloud customer. In these cases, the infrastructure belongs to one or several infrastructure providers from whom he purchases the needed resources. The service provider can purchase from multiple infrastructure providers to serve his own cause.
- *Infrastructure provider*: Infrastructure provider is the owner of the cloud physical infrastructure. This provider may also contract a network provider to use his service.

For instance, a media company wants to deploy a Online Video Streaming (OVS) service to serve its clients. It contracts a service development company to build and deploy the OVS service in the Internet. The service development company does not own the resources (e.g. virtual machines, storage volume, network links) but purchases them from GoGrid to deploy the OVS service in the latter's datacenters. In this example, the media company is the cloud customer who owns the OVS service. Its clients, subscribers of the OVS service, are the end-users who use the OVS service. The service development company is the service provider to whom the media company has to pay for building and deploying the OVS service. Finally, GoGrid is the infrastructure provider that the service development company has to pay for the resources it uses for deploying and running the service.

In this thesis, we position ourselves at the level of a cloud service provider. The mains is to develop new mechanisms to handle efficiently customers' requests. Since customers always want a solution that satisfies their requirements with as minimal cost, it is our objective to find the best solution in term of cost without compromising the quality of service.

2.8 General challenges

There are still a lot of issues that are not fully solved in Cloud Computing. In this section we will highlight some of the most significant challenges for the future of Cloud Computing.

2.8.1 Security issues

The first and the foremost concern of cloud customers is the security issue [128]. Since their data will be stored in the cloud, customers may have serious reasons to worry about. Not only they have to trust a third party for the protection of their data, but also customers could not locate where their data is stored. This increases their fears against uncertain factors such as regulatory regulations, disasters, conflicts, etc. Moreover, in addition to outside threats, the data inside the cloud is also a potential target of malicious co-tenants. Still, current cloud providers may offer adequate security protection for small and medium companies which do not possess sophisticated security solutions. Indeed, in a PwC's 2012 Global Information Security survey, 54% of the companies that had adopted the cloud said that the cloud actually improved their security system [125]. However, companies with high confidential data are not yet confident.

There are many threats in the cloud, which are classified in following categories (Table 2.1):

Threats (Referred in)	Description
Insecure interfaces and APIs ([37, 71, 87])	Users use API to communicate with the cloud and control their virtual machines and services. The transaction between the users and the API servers can be exploited to steal users' keys or access tokens, or circumvent policy.
Data leak ([37, 64, 69, 71, 76, 87])	Moving from corporate network to cloud requires additional security measure to prevent essential data from outside threat.
Cloud provider trust ([37, 64, 69, 71, 76])	Cloud provider has to ensure the customers of their data confidentiality, including data that customers store in the cloud and data collected by the cloud provider.
Malicious customers ([37, 71, 87])	Cloud provider must secure the cloud from possible attacks from malicious cloud customers. These attacks may try to compromise the hypervision layer to intrude to cloud's system or to other cloud customers' space.
Abused use of Cloud Computing ([71])	Malicious users, hackers and spammers can use public cloud to conduct their activities such as hosting malicious data, sending spams, retrieving stolen data, cracking password, launching denial of service attacks, commanding botnets, launching dynamic attack points, etc.
Account Hijacking ([37, 71, 87])	Like all online services, user's accounts and services are targets of hijacking. Phishing, fraud, exploitation of software vulnerabilities are common threats.
Auditability ([37, 64, 67, 71, 76])	It is hard for cloud customers to do audit in the cloud since they do not have the knowledge of underlying infrastructure. Even for cloud provider, the scale of the cloud make proper audit difficult.

Table 2.1: Security threats in Cloud Computing according to Cloud Security Alliance

1. Insecure interfaces and APIs: In cloud, customers interact with the cloud via its API, from requesting services to performing provisioning, management and monitoring. Therefore, protecting API transactions is critical to the cloud.

2. Data leak: Moving from private corporate infrastructure to the public cloud will need careful consideration about security. The security requirements must be adapted to the new network infrastructure. Some previously confined data are now exposed to Internet and shared in public network. Even with hybrid cloud where essential data is stored in the customers' premise, the private IT system is exposed to the public cloud, which constitutes a security threat. Thus Cloud Computing requires extensive access control and encryption to protect private data.
3. Cloud provider trust: In the cloud, data of millions of users may be hosted by a single cloud provider, and all have to trust the cloud provider to detain their data. Cloud providers have to ensure the confidentiality of their customers' data. Cloud provider's security policy must include human security measurement, e.g. preventing its employees from attempting malicious acts. Some cloud providers collect data from customers for management or improving customers' experience. These data, however, can reveal customers' information and be exploited for other activities with or without customers' awareness, e.g. advertising, spam or simply transferring to third-party service providers [16].
4. Malicious customers: In a multi-tenancy environment, many customers share the same resources. The security system consists of identity management, access control and isolating customers from other customers and from the hypervision system. There is a threat that a malicious customer tries to infiltrate into other customers' space or into the hypervision system. Some Cloud Computing systems use paravirtualization to improve the performance of guest virtual machines which may also become a security threat [96]. For instance, Rutkowska and her team have successfully compromised Xen using BluePill attack [116, 134]
5. Abused use of Cloud Computing: In public cloud, users can subscribe using only a valid credit card; some clouds offer a trial period. Malicious users, hackers and spammers can register and conduct their activities such as hosting malicious data, sending spams, retrieving stolen data, cracking password, launching denial of service attacks, commanding botnets, launching dynamic attack points, etc.
6. Account Hijacking: Like all online services, user's accounts and services are targets of hijacking. Phishing, fraud, exploitation of software vulnerabilities are common threats.
7. Auditability: Since customers have no control over underlying system, they could not know the detail of the system, its software version, code update, security design, etc. Customer have limited access to relevant network-level logs as well as ability to conduct thorough investigations and gather forensic data. Even for cloud provider, the scale of the cloud makes it hard to do proper audit. Some cloud providers use

resources from third party cloud providers, thus making auditing even more difficult.

2.8.2 Standardization and Interoperability

In an open world, customers would like to choose cloud providers they want, and moving their systems / apps to another when they feel like it without much of effort (portability). That is what happens to electricity nowadays: we can choose whichever electricity provider we want without changing our equipments, since they all have the same technical standards (same voltage, same frequency, although different from country to country). Alas, that's not yet the case for Cloud Computing. Current Cloud Computing offers are based on unique sets of architecture and API. Even toolkits whose purpose are to build private cloud compatible with public offers still lack compatibility to work with many public clouds. Most of them are based on Amazon REST-like API, that offers the possibility to move to Amazon EC2, but otherwise useless when moving to other public clouds such as Joyent or vCloud. Building system that expands from several public clouds, such as between Google AppEngine, Microsoft Azure, EngineYard, is almost impossible; and exporting systems/apps built in one public cloud to other public cloud (portability) is hardly possible. Some small cloud providers make their API compatible with big clouds in order to attract customers and compete with big cloud providers, but in overall application migration from one platform to another platform is exhausted manual labour. This leads to the problem of vendor lock-in: the choice of cloud provider is permanent, because migrating to another cloud provider is almost impossible. For example, force.com, although advertised as an open PaaS platform, is useful only for building salesforce.com applications; or Cordy Process Factory applications which are built from assembling other customized components, only work on Cordy's cloud.

There are several standardization efforts which aim to provide common interface and interoperability between clouds. Table 2.2 presents some of the most prominent ones. However, as S.Ortiz argued, the technology and market will need to mature and stabilize a bit before a widely accepted standard emerges [103]. The chief responsibility for ensuring the success of such a standard lies with the community of current and future customers of Cloud Computing - including enterprises both large and small [21]. For a standard to be widely adopted, it needs inputs from customers and cloud providers to cover all their needs. Some cloud customers and providers are gathering into groups such that The Open Cloud Manifesto [14], The Open Group [9] and Open Data Center Alliance [12] to provide use cases requirements and challenges to be addressed in standardization efforts.

Cloud standards	Organisations	Serialization	Protocol	Description
Cloud Data Management Interface (CDMI [118, 123])	Storage Networking Industry Association	JSON	RESTful HTTP	CDMI defines an interface for cloud storage. CDMI functionalities include create, retrieve, update and delete data, manage containers accounts, security access, monitoring and billing from the cloud.
Cloud Infrastructure Management Interface (CIMI [45])	Distributed Management Task Force (DMTF)	XML, JSON	RESTful HTTP	CIMI provides a standard interface for IaaS management and allows users' workloads to be moved from one cloud to another.
Open Cloud Computing Interface (OCCI [91, 92, 100])	Open Grid Forum	XML	RESTful HTTP	OCCI is a extensible API for IaaS cloud management. OCCI core defines 3 resource categories: Compute, Network, Storage; OCCI Infrastructure adds Network Interface and Storage Link.
Open Virtualization Format (OVF [44])	Distributed Management Task Force (DMTF)	XML	Any	OVF is an open extensible format for packaging and distribution of virtual appliances (virtual machines and software running on top of virtual machines).
Topology and Orchestration Specification for Cloud Applications (TOSCA [101])	Organization for the Advancement of Structured Information Standards (OASIS)	XML, WSDL	RESTful HTTP	TOSCA is designed to enable the interoperable description of application and infrastructure cloud services. TOSCA focuses on the relationships between components of a complex service, and the operational behaviour of these components (e.g., deploy, patch, shutdown).
Open Data Center Alliance Usage Model [12]	Open Data Center Alliance	Not defined	Not defined	Open Data Center Alliance Usage Model is a series of usage models to address challenges, requirements and guidelines in the cloud. It contains currently 8 usage models in four categories: Secure Federation, Automation, Common Management & Policy, and Transparency.

Table 2.2: Some cloud standards

2.8.3 Quality of service

According to a recent survey [128] of Micro Trend Inc, the quality of service, in particular the performance and the availability of cloud services, is the number two concern of cloud

customers, behind the security issue. Current cloud providers do not offer as high quality as does corporate computing. This makes it difficult for business companies to rely on. As service instances continues to scale up, the probability that an instance fails increases more and more. Cloud providers try to tackle the problem by providing fine grain monitoring data and failure recovery. However, VM restarting and migrating require several second to several minutes, which is unacceptable in current corporate computing.

The mass concentration of resources in a single cloud provider, and generally in one datacenter per region, makes it the single point of failure to the whole cloud. Because the cloud provider only uses one solution for their cloud service, what happens if this solution has some flaw (as every system has)? What happens if someone succeeds in hacking the system, or an incident or a disaster happens? The data of the cloud provider, of their customers, and of the clients of the latter may be lost. Thousands, millions users will be affected at the grandeur that we have never seen before. On January 2^{nd}, 2010, a routing device caused Amazon EC2 outage affecting many sites and platforms that were hosted in Virginia datacenter. Among those, Heroku's 22 virtual machines suddenly vanished, resulting in estimated 44,000 running applications in the lurch. It was not the first outage at Amazon's datacenters. In 2008 there are also two outages in S3 services due to the authentication service overload and a single bit error. The same year, Google AppEngine also had 5 hours of partial outage due to a programming error [18].

Cloud providers use Service Level Agreement (SLA) to formally document the quality of service (QoS), performance expectations, responsibilities and limits between cloud providers and their customers. A typical SLA describes QoS levels using various attributes such as service availability, data durability, respond time, and penalties associated with violations of such attributes. In its report [78], the NIST concludes that at the moment, "an industry-wide accepted standard SLA form for cloud services does not exist", and "public cloud SLAs in their current form are of little value to customers". The SLA must be designed properly in order to reduce potential conflict between service providers and cloud customers. Once the SLA is defined, service providers have to be capable of measuring and monitoring relevant metrics in fine grains. But often there is a gap between correlating the collected metrics and higher-level functional guarantees. This problem is quite challenging and service management has to take this into account [88].

2.8.4 Slow data transfer

While storage capacity is cheap and easy to acquire, link resources are not so fortunate. Even if we can install more fibres to increase bandwidth capacity, achieving overall high

Internet speed and low cost is still far away. Imagine how long it would take to transfer the data of a company to the cloud, or moving a company's data from one datacenter to another datacenter.

Jim Gray, head of Microsoft's Bay Area Research Center, found that shipping the whole data disks or computers are a lot quicker than transferring data over Internet [18]. Let's take an example: assume that we want to ship 10TB from U.C. Berkeley to Amazon in Seattle, WA. The writing bandwidth is measured to be 5-18Mb/s. Let's make it 20Mb/s, the transfer of 10TB data would take as much as 4,000,000 seconds, equivalent to 45 days! If we ship the disk, then it would take less than a day to transfer 10TB, equivalent to 1,500Mb/s! Amazon currently proposes such a service called AWS Import/Export that ships the storage device to Amazon datacenter [2].

2.9 What's next?

Current cloud market is fragmentary and dominated by giant actors such as Amazon, Google and Microsoft. Lack of cloud standard and interoperability, current customers have no other choice but to choose and stick with one cloud provider that is most suitable for their needs. However, once Cloud Computing is widely adopted, future customers will be more demanding in terms of quality and range of services. Thus, it is expected that customers requests will be more complex and therefore harder to satisfy by isolated cloud providers, even the largest ones. Cloud experts expect future cloud providers to cooperate in order to better serve the customers' needs and create more opportunities [106, 125]. Hence, the future market place is not only a place for customers to meet cloud providers, but also for cloud providers to meet each other (Business-2-Business). And with the needs for cooperation between cloud providers, the incompatibility barrier will be addressed facilitating interoperability and partnership.

In this vision the cloud providers will form partnership to complement their services, which eventually leads to the appearance of the *cloud federation*. Cloud federation is a union of two or more cloud providers who join together to create a *federated cloud* [113]. The federation members share part of their resources for a price so that other members in needs of resources can purchase it. This sharing of resources will benefit both cloud providers and cloud customers. Each cloud provider in a federation can easily request resources from another member of the federation to satisfy customers' requests that are beyond his capability. He can also scale-out his datacenters with external resources, share and aggregate resources with other members' infrastructures to increase the computing capacity and re-

duce costs [94]. For cloud customers, they can choose a variety of providers selecting the most suitable services the market has to offer [75].

To get the full benefit from cloud federation, the orchestration mechanism would be challenging. Unlike current clouds where all datacenters belong to a single cloud provider, thus provide the same services, quality and price, a cloud federation is a heterogeneous environment where each datacenter provides a set of services with their own characteristics, quality and prices. Therefore, service deployment mechanism needs to take into consideration this heterogeneity and handle it efficiently. The objective of this thesis is a contribution to the resolution of this challenging providing efficient mechanisms for service deployment in a federated environment.

2.10 Conclusion

Although the concept is still new, it is agreed that Cloud Computing is the future of IT as the Internet is for communications. Indeed, it follows the rational externalization process that already happened in industry. Cloud Computing transfers the IT-related responsibility from customers to cloud providers, allowing them to focus on their core business. By concentrating massive capital, resources and technologies in the cloud providers, Cloud Computing business model allows reducing the overall IT cost and waste of resources, efforts as well as risks. It converts Capital Expenditure (CAPEX) to Operating Expenditure (OPEX), eliminating up-front investment that blocks the small and start-up companies. Customers can request as much resources as needed and pay only for what is used. Therefore, they do not need to provision far ahead for their services. They can build, test, deploy and commercial their services while letting the cloud providers handling all the background work. However, even with many cloud offers already in the market, Cloud Computing is still facing a lot of challenges, such as security, standardization, quality of service, and slow data transfer. Industrial and academic are working hard to address these challenges. This thesis is one contribution to the challenges that address specifically the problem of service deployment in the cloud.

Part II

Contributions

Researchers already see a future where Cloud Computing is no longer a fragmentary environment where customers are locked-in with one Cloud Computing provider. Next-generation cloud providers are expected to be open, capable to join together to share their resources as a common cloud infrastructure to provide wider range of service offers. It is essential for all cloud providers to be able to extend their offers beyond the boundary of their physical infrastructure (datacenters) or logical one. Even Amazon AWS, the foremost and by far the largest IaaS cloud [13], lacks many features that business customers desire, such as the support for SLA and quality of business level. Cooperation with other cloud providers may open services that they couldn't offer to customers. With this vision, arises the need to support interoperability between clouds and service deployment over heterogeneous cloud environment. In such an environment, each cloud proposes its own service catalogue, its own service quality and price. For customers who want to deploy complex services, such cloud makes manually calculating and deploying provisioning plan tiresome and time-consuming. Therefore, automatically provisioning services over the cloud spreading different locations will be a powerful tool for service providers to support their customers. Supporting complex service deployment is the purpose of our contributions.

Chapter 3

Efficient and effective service composition in cloud infrastructure

3.1 Introduction

With many actors from industry, cloud computing is ready to serve the customers' needs for on-demand computing resources. However, third party services are not going to develop as quickly as envisaged except for raw resources requests such as VMs or hosting environment. Indeed, customers can request development environment from PaaS provider, but they still need to code the services as they did years before. For companies that want to deploy a service (for their employees/partners/clients, etc), the cost for building the service may be higher than the cost of the raw resources it consumes and constitute an important part of the companies' IT budget.

In our approach, we envision that cloud service providers would be able to provide services as quickly as they do with raw resources now. They should be able to compose new services from already available service components to speed up the service delivery. The idea is to allow them to easily and cost-effectively deploy complex portfolio of services directly into the infrastructure of the resources providers (IaaS) and invoke them upon customer demands. We call the capability of composing already deployed services into a new complex one the *"service composition"*. With this capability, service providers can offer quickly wider range of services based on already deployed basic services in the network, making new business faster to set up. Compared to the current PaaS model which provides only development environment, this model allows reducing service deployment with little effort from service providers, thus decreasing service building cost. Moreover, it also allows breaking down complex service into pieces which are provided from different service

providers, encouraging innovation.

There are some primitive initiations from Force.com [117], Cordys Process Factory [42] and Rollbase [130] who provide customers with tools to create new applications without using development environment. Instead, they provide a lists for pre-built customizable components so that customers can integrate them together to form a new application as their need. Thus the services can be quickly set-up and run with little effort from customers and service providers. However, these are still simple components destined for web service-like applications such as forms, email, or tables.

Some current research projects such as CompatibleOne [40] and Resevoir [112, 114] provide mechanisms for service providers and customers to request raw resources (virtual machines, storage volumes) from different cloud infrastructures. Customers can request raw resources and make simple configuration for their disassembled service components in different clouds. This grants service providers and customers the possibility of choosing services from various cloud infrastructure providers that they see most suited for their demand.

To get the full benefit from service composition capability, service orchestration would be the main challenge [135]. Indeed, with the move of data and workload from computers to the cloud, requests for data storage and services will massively increase in number and variety. customers' requests may vary from long time service that results in a several-year contract to short time service that lasts for several minutes. As a result, there is a clear need for orchestration mechanisms that are capable of deploying complex services in an optimal way in term of computing resource usage and operating cost in brief delay. This is the purpose of our first contribution.

Chapter structure

This chapter is structured as follow. In Section 3.2 we present the problem statement. Then in Section 3.3 we introduce most relevant works in this topic. Section 3.4 presents our graph-based model of the service composition problem. In the Sections 3.5, 3.6 and 3.7 sections we provide analysis and algorithms for service provider to deploy a complex service with the minimal cost. Section 3.5 proposes an algorithm for a complex service in form of a path, while Section 3.6 deals with services in form of star topology (hub & spoke), and Section 3.7 deals with services of tree topology. The three proposed algorithms are, in fact, the same algorithm presented in Section 3.7 but broken down into three topologies from simple to more complicated for the sake of better presentation. Section 3.8 and 3.9 dedicate to the problem where service graph is of general topology. We will provide our analysis of the problem in this case, propose a solution for this problem and evaluate it

through simulation. Finally, we conclude the chapter in section 3.10.

3.2 Problem statement

Let's consider a scenario in which a media company wants to deploy an Online Video Streaming service destined to the French market. The request of this company to a cloud provider is a complex service which is composed of a Storage node to stock the media content, a Transcode node to adapt the media to its users' terminal supports, and a Portal node as detailed in Table 3.1. Since the videos are converted in Transcode node and streamed directly to users' terminals in France, the company requires that the Transcode node is also located in France. This makes it easier to ensure the quality of the service delivered to the end-users. The Storage node does not have constraint on location, since users do not have direct connection to it, but via the Transcode node. However, it requires very high availability and an important traffic link to the Transcode node. Finally, the Portal node serves as users' access and control node. It does not requires either location constraint nor high availability but needs only a low traffic link to Transcode node. Two instances of this node are required for better redundancy.

Request components	Quantity	Requirements / Attributes
Storage Node	1,000 Gb	Availability :98%
Transcode Node	10 instances	Location : "eu:france" Availability : 95%
Portal Node	2 instances	None
Link Storage - Transcode	50 Gb/hour	None
Link Portal - Transcode	5 Gb/hour	None

Table 3.1: Online Video Streaming Request

Instead of setting up the whole service from scratch, the cloud service provider will try to take benefit of already installed basic services, thus effectively reducing the cost and operation time. From the perspective of the cloud service provider, to fulfil this request, it needs to set up a mechanism to: 1) identify the basic services that compose the required complex service, and 2) select which service instances (and corresponding hosts) to activate in the cloud to build the required complex service logic.

In this work, the complex service is modelled as an abstract graph (called *service graph* or *request graph*) where each node indicates the required basic service. This approach indeed simplifies the identification of basic services. The *service composition problem* (or *complex service mapping problem*) is therefore the problem that, given the basic service

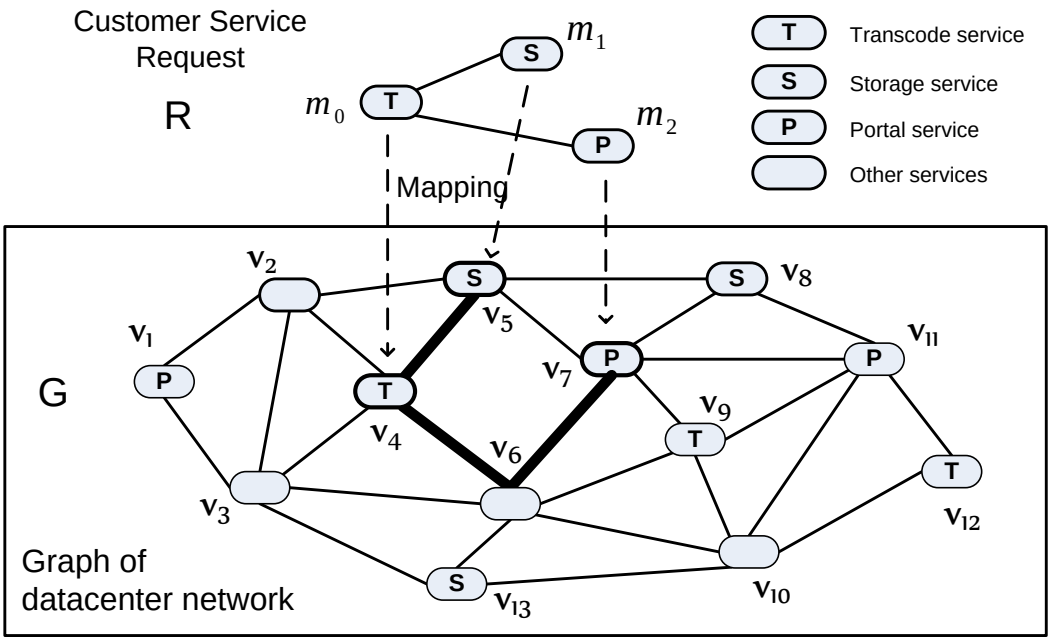


Figure 3.1: Mapping Online Video Streaming service graph R into substrate network graph G

nodes already deployed in a substrate network, how to optimally select a set of service nodes and links among them to which the abstract service graph can be mapped (Fig. 3.1). In this chapter the Service Level Agreement (SLA) is simplified and only specifies the required resources (computing & networking resources). We suppose that the network resources are very large compared to the users' demands so that the capacity is not a constraint. We will provide an analysis of the problem and propose solutions for different request complex service graph topologies, varying from path graph to tree.

3.3 Related works

There are many researches on service composition [115, 137]. Most of them address the problem of service matching while assuring QoS [24]. In our work, the complex service is explicitly expressed in term of service description, that is known to the substrate nodes. In fact, if the explicit service expression is unknown, one may find many combinations of basic services that produce the same result [97]. To reduce the problem, we require that complex services are explicitly represented as abstract graphs where each service associated with each node is well described and service matching mechanism is provided.

In [35, 36], Choi and Tunner presented an algorithm to solve the problem where the service graph is a path. They built a multi-layered graph, each layer corresponding to a basic

service in the service path, then used Dijkstra algorithm to find the minimal path. The algorithm runs in time O(Ek) where E is the number of edges in the substrate network, and k is the number of required services in the path. They mentioned that the problem is NP-hard in case of limited link capacity. Raman and Katz [108] used this algorithm to achieve load balancing for path-topological service composition. To this end, they defined a metric with low cost in favour of nodes and links with higher capacity, then calculated the minimal service composition. In [65], Gu et al. used a similar strategy to find an approximately optimal service path under QoS constraint. They also built the graph using new cost metric which was calculated based on QoS criteria. They also applied Dijkstra algorithm to find the minimal service path.

In [80, 81], the authors proved that the problem is NP-hard if node capacity constraint exists. It is because several abstract nodes can be mapped to a same substrate node, thus the cumulative required resource could exceed the node capacity. The authors proposed a greedy heuristic algorithm which, starting from the source node, tried to reach the destination of the path service. At each step, it consumes the required resources for the service from the current substrate node, or moves to one of its neighbours if the resources are not sufficient. While the algorithm can successfully reach the destination (i.e. produce a solution) in only 94% of time, the simulation suggests that the solution performance is very close to the optimal.

Wang et al. [131] proposed an $O(V^3)$-runtime algorithm that used Wang-Crowcroft algorithm to compute the path service. They extended it into an $O(V^3)$-runtime approximation algorithm to solve the case where the complex service is a directed acyclic graph (DAG) by gradually splitting the DAG into disjoint paths.

Cai et al. [27] only took node resource in their objective function. They addressed the problem of re-mapping abstract graph in case of changes of the substrate graph. Their greedy algorithm gradually picks up the node whose constraint cover set is the biggest (a constraint cover set consists of all the nodes whose link to the picked node does not violate the constraints). Using simulation, they showed that their solution is better than random algorithm and close to optimal.

Another popular strategy is to present the problem in term of multi-commodity flow. Using this method, Liang et al. [79] solved the problem of VPN-embedding with star topology (hub-and-spoke). Lately, by introducing meta-nodes, Chowdhury et al. [39] modelled the problem in a form of multi flows between meta-nodes. They then solve the Mix Integer Programming associated to the problem using Relaxation and Rounding method. The algorithm can be used to solve simultaneously multiple requests. However, the size of the equation system increases with the number of requests, so does the complexity.

$G = (V, E)$	Substrate network graph
$R = (V^R, E^R)$	Requested service graph
m, n, m_i	Abstract nodes
$a, mn, m_i m_j$	Abstract edges
$r(.)$	Demanded resources (node & link resources)
u, v, u_i, v_i	Substrate nodes
$x, uv, u_i u_j$	Substrate edges
$s(.)$	$s : V \cup V^R \rightarrow S$ Service function
$P(.)$	$P(m)$: set of candidates of m
$\mu(.)$	Cost (per resources unit)
$\Phi = (\Phi_V, \Phi_E)$	Map function; Φ_V, Φ_E maps on node & edge sets
$f(.)$	Cost function

Table 3.2: Notations used in this chapter

Most of these works limit themselves to complex services in a particular topology, we propose a solution for all types of requests in the form of path, star (hub-and-spokes), tree and general graph topologies. We first present the TRM algorithm that can find the optimal solution for service graph of path, star and tree-topology in time $O(kV^2)$ with preliminary calculations, k being the number of abstract nodes and V being set of substrate nodes. If all required basic services are distinguishable, the algorithm terminates in time $O(V^2)$. Finally we propose its extension, IGM, which is capable of solving the problem without any restriction of service graph topology. Due to the hardness of general graph topology, we will discuss this case separately in Session 3.8.

3.4 Mapping complex service into a substrate network

In this section we will introduce our model of the complex service mapping problem. In our model, a substrate network is a graph where each node provides a particular service. Using the resource on any node or any edge incurs a certain cost per resource unit.

A complex service request is represented as an abstract graph in which each abstract node indicates the service and the quantity of computing resources it requires and each edge indicates the quantity of networking resources it requires. For instances, consider the Online Video Streaming example mentioned in the Introduction. The required complex service consists of a Portal Server, a Storage Server and a Transcode Server and two links which represent the communication flow among these servers (Fig. 3.1). The resources requirement for the Storage Server is expressed in term of storage capacity (Gb). For the Transcode Server and Portal Server, the resources requirements are expressed in term of

number of instances (at IaaS level, an instance is a virtual machine, i.e. a pre-built virtual host in which CPU, RAM and disk storage are defined). Finally, the resources requirement for network connection between servers is in term of traffic rate (e.g. GB per hour).

Our objective is to map the complex service graph to the substrate network graph (abstract node to substrate node that provides the required service; abstract edge to a substrate path connecting the substrate nodes) while minimizing the usage cost of underlying substrate resources.

Substrate network graph

Let S be a set of basic services. Let $G = (V,E)$ be a substrate graph. Each node $u \in V$ is able to provide a service $s(u) \in S$ at usage cost $\mu(u)$ per used resource unit (e.g. per time CPU, per Gb of storage, etc.). Each edge $x \in E$ has a cost $\mu(x)$ per used resource unit (e.g. per Mb of bandwidth, per Gb of traffic throughput, etc.). From this definition, we define the usage cost per required resource unit of any path p in G as: $\mu(p) = \sum_{x \in E \cap p} \mu(x)$.

Complex service request

A complex service requested by a customer is represented as an abstract graph $R = (V^R, E^R)$. Each abstract node $m \in V^R$ represents a required basic service $s(m) \in S$ with its associated required node resource $r(m)$. Each abstract edge $a \in E^R$ represents a required link resource $r(a)$.

Service mapping mechanism

The objective of the service mapping mechanism is to find a set of substrate nodes and edges that satisfy the requirements of the complex service request. With this mapping, each abstract node will be associated with an underlying substrate node that provides the corresponding service and satisfies the resource requirement. In addition, communication links between any pair of these substrate nodes must satisfy the resource requirement of the abstract edge between the correspondent abstract nodes (Fig. 3.1). We specify such a mechanism by a mapping function (or a *map*) $\Phi = (\Phi_V, \Phi_E)$ which is composed of two sub-functions (respectively vertex map and edge map) $\Phi_V : V^R \to V$ and $\Phi_E : E^R \to \mathcal{P}(E)$, $\mathcal{P}(E)$ being set of all subsets of E, such that:

$$\begin{cases} \Phi_V : m \in V^R \longmapsto u \in V \text{ such that } s(u) = s(m). \\ \Phi_E : a = mn \in E^R \longmapsto p, \quad p \text{ being a path of G linking } \Phi_V(m) \text{ and } \Phi_V(n). \end{cases} \tag{3.1}$$

For any $m \in V^R$, we call $u = \Phi_V(m)$ the *target* of m by the map Φ. The cost of the structure, which is called the "mapping cost of Φ", is calculated by summing up all the resource usage costs in G (i.e. node and link usage costs):

$$f(\Phi) = \sum_{m \in V^R} \mu(\Phi_V(m)) r(m) + \sum_{a \in E^R} \mu(\Phi_E(a)) r(a)$$

Let's consider the Online Video Streaming service example presented in Section 3.1. Suppose that the fees of the cloud provider are: \$0.001/Gb per hour for any Storage server; \$0.30 per any Transcode instance of (2GHz CPU + 1 Gb RAM) per hour; the cost of a portal instance of (2GHz CPU + 1 Gb RAM) is \$0.50 per instance per hour and \$0.10 per Gb per hour for any link.

If using the map illustrated in Fig. 3.1 (remark that the abstract edge m_0m_2 is mapped onto a substrate path of 2 substrate edges), deploying the Online Video Streaming service will result a monthly cost:

$$\begin{aligned} f(\Phi) &= \$0.001 \times 10000 + \$0.30 \times 10 + \$0.50 \times 2 + \$0.10 \times 50 + (\$0.10 + \$0.10) \times 5 \\ &= \$20.00 \text{ per hour} \end{aligned}$$

This example is simple since all instances have the same size and node and links resource price depends only on their provided service. The summary of the example is shown in Table 3.3.

G	Storage hosts	\$0.001/Gb per hour
	Transcode hosts	\$0.30 per instance per hour
	Portal hosts	\$0.50 per instance per hour
	Link	\$0.10/Gb per hour
R	1 Storage Node	10,000 Gb
	1 Transcode Node	10 instances
	1 Portal Node	2 instances
	Link Storage Node - Transcode Node	50Gb per hour
	Link Portal Node - Transcode Node	5Gb per hour
Map	Φ as illustrated in Fig. 3.1	
Cost	$f(\Phi) = 0.001 \times 10000 + 0.30 \times 10 + 0.50 \times 2 + 0.10 \times 50 + (0.10 + 0.10) \times 5 = 20.00\$$ per hour	

Table 3.3: Example of Online Video service mapping cost

Optimal service mapping problem

The optimal service mapping problem aims to find an optimal map Φ that minimizes the cost $f(\Phi)$. Since the number of possibilities of mapping the graph R to G is limited, the minimal mapping cost of R always exists. We denote $f_{min}(R)$ as the minimal mapping cost

of R.

Definition 3.1. *Complex Service Mapping Problem.*
Given a network graph $G = (V, E, S)$ and a request $R = (V^R, E^R, S)$, find the minimal map Φ from R to G, i.e. finding the map $\Phi = (\Phi_V, \Phi_E)$ from R to G which minimizes:

$$f(\Phi) = \sum_{m \in V^R} \mu(\Phi_V(m)) r(m) + \sum_{a \in E^R} \mu(\Phi_E(a)) r(a) \tag{3.2}$$

Suppose that $\Phi_V(m_i)$ is determined for all $m_i \in V^R$, then the optimal mapping of any abstract edge $m_1 m_2 \in E^R$ is indeed the minimal path (shortest path) which links the substrate nodes $\Phi_V(m_1)$ and $\Phi_V(m_2)$. Therefore, we need only to consider the mapping in which all edges are determined by the minimal path between its two ends to solve the problem. Henceforth, any mapping in this work is well identified by its selected substrate nodes. If we denote by $\mathcal{D}(u,v)$ the minimal substrate path between $u, v \in V$ and $d(u,v)$ its length (cost per resource unit), then $\Phi_E(m_1 m_2) = \mathcal{D}(u_1, u_2)$, given u_1 and u_2 the substrate nodes of m_1 and m_2. Using Floyd-Warshall algorithm [54], we can calculate all these paths in time $O(V^3)$.

Therefore, we deduce from (3.2) the following expression:

$$f(\Phi) = \sum_{m \in V^R} \mu(\Phi_V(m)) r(m) + \sum_{m_i m_j \in E^R} d(\Phi_V(m_i), \Phi_V(m_j)) r(m_i m_j) \tag{3.3}$$

Since we can derive the map Φ from Φ_V, we will use Φ at the place of Φ_V in the remaining of the chapter for the the purpose of clarity of the notation; $\Phi(m)$ will present the target of m with the map Φ.

Service matching assumption

For any abstract node $m \in V^R$, let $P(m)$ be the set of all substrate nodes which are able to provide the instances of the same service as required by m. A member of $P(m)$ is called a *candidate* of m. In practice, a “service" is described by a set of functions (functionalities proposed by the service) and attributes (possible parameters of these functions). For any $m \in V^R$, $P(m)$ is the set of all substrate nodes which match the services described by the abstract node. Therefore, the definition of S is abstract, and the mapping mechanism will only consider the substrate nodes that provide services matching the abstract nodes' service description.

As an example, Fig. 3.2.a illustrates a simple JSON schema [138] for describing a service.

```
{
    "type": "object",
    "properties": {
        "service" : {
            "type" : "object",
            "properties": {
                "name": {
                    "type": "string",
                    "required": true
                },
                "function": {
                    "type": "string",
                    "required": true
                },
                "attributes": {
                    "type": "array",
                    "items": {
                        "type": "object",
                        "properties": {
                            "key": {"type": "string"},
                            "value": {"type": ["number","string","object"]}
                        }
                    }
                }
            }
        }
    }
}
```

(a)

```
{
    "service": {
        "name": "Service_1",
        "function": "transcoding",
        "attributes": [
            {"key": "location", "value" : "EU:France"},
            {"key": "accepted_format", "value": ["MPEG2","MPEG4"]}
        ]
    }
}
```

(b)

```
{
    "service": {
        "name": "Service_2",
        "function": "transcoding",
        "attributes": [
            {"key": "location", "value" : "EU:France:Paris"},
            {"key": "accepted_format", "value": ["MPEG2","MPEG4","H.263"]}
        ]
    }
}
```

(c)

Figure 3.2: Example of Transcode services description. (a): Service schema; (b): Transcode service 1; (c): Transcode service 2

The main tag is “service", and it must contain 2 fields “name" and “function" of type string. The appearance of these fields are imperative (“required" is set to true). It may contains an array “attributes" in which each item is a couple (“key", “value"). From this schema we define two simple transcoding services specification with different accepted formats (i.e. MPEG2, MPEG4, and H263) which all are deployed in France as specified in Fig 3.2. In this example any node that provides *Service_2* will match the service description of *Service_1*.

This example shows that it is possible that a service provided by a substrate node can match more than one abstract nodes' service description. Therefore, the expression “$s(u) = s(m)$" should be understood that $s(u)$ matches the service description $s(m)$. For the sake of simplification, we keep the expression “$s(u) = s(m)$" in the remaining of the chapter. In this chapter we suppose that there is at least one substrate node that matches the service required by each abstract node, otherwise the request will simply be rejected.

Unlimited capacity assumption

One should notice that using the shortest path between two substrate nodes in graph G as the mapping of the abstract edge between their corresponding abstract nodes is not always feasible if there is a capacity constraint. Under capacity constraint, the complex service mapping problem is NP-hard [35], avoiding the possibility to find the optimal solution in polynomial time. In our model, we relax the capacity constraint, given that the cloud context is specific. Indeed, contrary to Virtual Network Embedding (VNE) problem [39] where the physical network capacity is limited, cloud service providers can always request more underlying resources from IaaS providers since this is part of the concept of the cloud. In cloud business model, cloud service providers are simply (cloud) customers of infrastructure providers. Cloud Computing technology aims to transfer the provisioning responsibility from cloud service providers to IaaS providers: cloud service providers do not have to provision physical resources far ahead to provide services for their future users, but rather provision what is actually needed, then scale up whenever needed. This can be done proactively so that they can never run low of resources. Many estimation and prediction techniques such as PRESS [63], MBRP [48], CloudScale [119], Gmach's automated method [59] or Chandra's techniques [31] could help service provider anticipate the needs for additional resources.

For instances, Heroku [1] provides Ruby, Java and Python development platform (Platform as a Service) to its customers. Heroku does not own any physical infrastructure but host its customers' services on top of Amazon EC2 infrastructure. Heroku is a cloud customer of

[1] http://www.heroku.com

Amazon. It purchases resources from Amazon to extend its platform whenever needed (e.g. new customer requests). Amazon provides resources in the form of virtual machines (VM). Heroku may, at any time, request additional VMs or release the instantiated ones. More importantly, Heroku pays only for the used VMs; there is no commitment or preliminary contract. IaaS providers such as Amazon invest into massive quantity of resources to make sure that their customers' demands are always satisfied. This is a crucial aspect of Cloud Computing; and therefore, from the viewpoint of cloud service providers such as Heroku, there is no limit of resources they can purchase from the IaaS providers, as if they draw from an infinite pool of resources: this is called in the literature the "appearance of unlimited resource capacity" [18], or "illusion of unlimited resources capacity" [126]. Therefore, it is reasonable in this research to assume that the network capacity is unlimited.

3.5 The path-topological complex service

In this section, we consider the graph R of a path topology. Precisely, $V^R = \{m_0, m_1, ..., m_{k-1}\}$ and $E^R = \{m_0m_1, m_1m_2, ..., m_{k-2}m_{k-1}\}$. We first ignore all the nodes which services are not required in the abstract graph R. This assures that any remaining node is eligible to be used in the mapping. Denote the set of all remaining substrate node by V'. For every abstract node, we creates a subset of V regrouping all substrate nodes that provide the same service required by the abstract node:

$$V' = \bigcup_{i=0}^{k-1} V_i,$$

where V_i = set of all substrate nodes u_i in V' which provide service $s(u_i) = s(m_i)$.

It is trivial that $\Phi(m_i) \in V_i \quad \forall i = 0..k-1$. The problem of finding Φ is equivalent to find $u_0, u_1, ..., u_{k-1} \in V_0, V_1, ..., V_{k-1}$ respectively so that the mapping cost of R is minimized:

$$\begin{aligned} &Min\ f(\{u_0, u_1, ..., u_{k-1}\}), \\ &f(\{u_0, u_1, ..., u_{k-1}\}) = \textstyle\sum_{i=0}^{k-1} \mu(u_i) r(m_i) + \sum_{i=0}^{k-2} d(u_i, u_{i+1}) r(m_i m_{i+1}) \end{aligned} \tag{3.4}$$

We solve this case by recurrence. Let R_i be the sub-graph of R restricted in $\{m_i, m_{i+1}, ..., m_{k-1}\}$ for i = 0,1,..,k-1. Then $R_0 = R$ and R_{k-1} = singleton $\{m_{k-1}\}$. Let $f_{min}(R_i, u_i)$ be the minimal value of mapping cost of R_i if the node $u_i \in V_i$ is chosen (i.e. the minimal mapping cost among all the possible mappings of R_i to G where m_i is mapped to u_i), then:

$$f_{min}(R_i) = \min_{\forall u_i \in V_i} f_{min}(R_i, u_i)$$

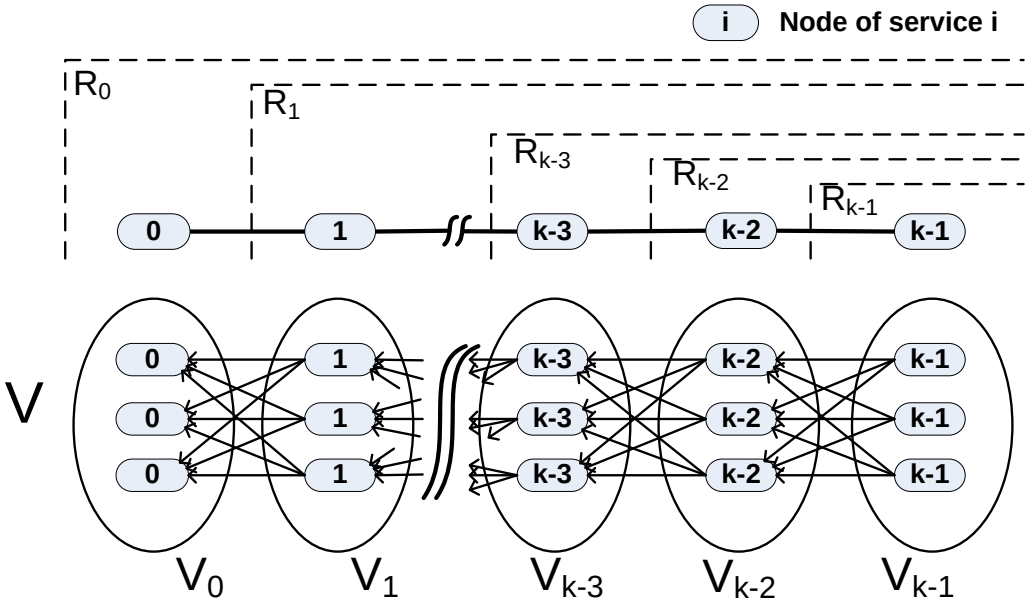


Figure 3.3: Minimal mapping cost with path graph request scheme

Let consider the graph R_i where a node u_i is chosen. One can remark that R_i is composed of R_{i+1} in addition to abstract node m_i and abstract edge $m_i m_{i+1}$. So if we choose a node $u_{i+1} \in V_{i+1}$ as the host of m_{i+1}, then the best choice for the rest of R_i is the best solution of mapping R_{i+1} given that u_{i+1} is chosen. Hence we have:

$$f_{min}(R_i, u_i) = \min_{u_{i+1} \in V_{i+1}} (f_{min}(R_{i+1}, u_{i+1}) + \mu(u_i) r(m_i) + d(u_i, u_{i+1}) r(m_i m_{i+1})) \tag{3.5}$$

From this analysis, we can calculate f_{min} at any node of service i from the value f_{min} of all the nodes of service i+1 (Fig. 3.3), knowing that f_{min} at each node u_{k-1} of service k-1 is: $f_{min}(R_{k-1}, u_{k-1}) = \mu(u_{k-1}) r(m_{k-1})$ (Algorithm 1).

The validity of Algorithm 3 is proved by the following proposition:

Proposition 3.1. *For all i = 0,1,2..,k-1, for all $u_i \in V_i$, the value $f_{min}(R_i, u_i)$ calculated by the Algorithm 1 is the minimal cost of all the map of R_i into graph G in which m_i mapped to u_i.*

Proof. We prove the proposition by recurrence by i, starting from k-1 down to 0. It is trivial that the proposition holds for i=k-1, since there is only one possible map from R_{k-1} into G in which m_{k-1} is mapped to u_{k-1}.

Suppose that the proposition holds for i, $i \leq k-1$. We prove that the proposition also

Algorithm 1 Complex service mapping for path graph

Input substrate graph G = (V,E,S), unity cost μ, abstract graph R = $(V^R, E^R, S), V^R = \{m_0, m_1, ..., m_{k-1}\}, E^R = \{m_0m_1, m_1m_2, ..., m_{k-2}m_{k-1}\}$.

Output minimal mapping cost $f_{min}(R)$

1: V_i = all substrate nodes of service $s(m_i)$ for i=0,1,..,k-1.
2: Calculate $d(u_i, u_{i+1}) \quad \forall i = 0..k-2 \quad \forall u_i \in V_i, \forall u_{i+1} \in V_{i+1}$.
3: $f_{min}(R_{k-1}, u_{k-1}) = \mu(u_{k-1})r(m_{k-1}) \quad \forall u_{k-1} \in V_{k-1}$.
4: For i = k-2 downto 0 do
    For all $u_i \in V_i$ do
        $f_{min}(R_i, u_i) = \min_{u_{i+1} \in V_{i+1}}(\mu(u_i)r(m_i) + d(u_i, u_{i+1}))r(m_im_{i+1}) + f_{min}(R_{i+1}, u_{i+1}))$.
5: $f_{min}(R) = \min_{u_0 \in V_0} f_{min}(R_0, u_0)$.

holds for i-1. Consider the map, which we denote by Φ_{i-1}, of R_{i-1} into G in which m_{i-1} is mapped to u_{i-1} and the cost is minimal. Suppose that u_i is the substrate node to which m_i is mapped by Φ_{i-1} . Then, following the recurrent hypothesis, the cost of mapping the part $m_im_{i+1}...m_{k-1}$, or R_i into G, denoted by Φ_i, is at least $f_{min}(R_i, u_i)$: $f(\Phi_i) \geq f_{min}(R_i, u_i)$.

Thus the mapping cost of Φ_{i-1} is:

$$\begin{aligned}
f(\Phi_{i-1}) &= \sum_{m \in V^{R_{i-1}}} \mu(\Phi_{i-1}(m))r(m) + \sum_{a \in E^{R_{i-1}}} \mu(\Phi_{i-1}(a))r(a) \\
&= \mu(\Phi_{i-1}(\Phi_{i-1})r(m_{i-1}) + \mu(\Phi_{i-1}(m_{i-1}m_i))r(m_{i-1}m_i) + \\
&\quad + \sum_{m \in V^{R_i}} \mu(\Phi_{i-1}(m)r(m) + \sum_{a \in E^{R_i}} \mu(\Phi_{i-1}(a))r(a) \\
&= \mu(u_{i-1})r(m_{i-1}) + d(u_{i-1}, u_i)r(m_{i-1}m_i) + f(\Phi_i) \\
&\geq \mu(u_{i-1})r(m_{i-1}) + d(u_{i-1}, u_i)r(m_{i-1}m_i) + f_{min}(R_i, u_i) \\
\Rightarrow f(\Phi_{i-1}) &\geq f_{min}(R_{i-1}, u_{i-1})
\end{aligned}$$

By the definition of Φ_i, $f_{min}(R_{i-1}, u_{i-1})$ must be greater than or equal to $f(\Phi_{i-1})$. Consequently, we conclude that $f_{min}(R_{i-1}, u_{i-1}) = f(\Phi_{i-1})$ the minimal mapping cost of R_{i-1} given that m_{i-1} is mapped to u_{i-1}. □

The proposition proves that for any $u_0 \in V_0$, $f_{min}(R_0, u_0)$ is the minimal cost of all the maps that map m_0 to u_0. Thus the minimal cost of all the mapping is taken among $f_{min}(R, u_0), u_0 \in V_0$.

The most time-consuming step in the Algorithm 1 is the step 2 where the minimal paths are processed. The Floyd_Warshall algorithm can calculate the distance between all pairs of nodes in time $O(V^3)$. In the case where each substrate node is capable to perform some

computing (e.g. distributed network), then all these calculations can be done in parallel at each node using Dijkstra algorithm, thus reducing the complexity by a factor V. If this step can be calculated beforehand, then the remaining of the algorithm runs in time:

$$O(\sum_{i=0}^{k-1} V_i V_{i+1}) = O(kV^2)$$

If the required services in the service graph are distinguishable (i.e. V_i are disjoint), then the left expression is equivalent to $O(V^2)$.

3.6 The star complex service

We consider a graph service R which has a star topology (or hub-and-spoke topology). $R = (V^R, E^R, S)$ where $V^R = \{m_0, m_1, ..., m_{k-1}\}$ and $E^R = \{m_0m_1, m_0m_2, ..., m_0m_{k-1}\}$.

Again, we divide V' into subsets $V_0, V_1, ..., V_{k-1}$ where every node in V_i has the same service as m_i for i = 0,1,..,k-1. If a $u_0 \in V_0$ is chosen as the substrate node of m_0, then the best choice of substrate node for any m_i is $argmin_{u_i \in V_i}\{d(u_0,u_i)r(m_0m_i) + \mu(u_i)r(m_i)\}$. Therefore, the minimal mapping cost if u_0 is chosen as the substrate node of m_0 is:

$$f_{min}(R,u_0) = \mu(u_0)r(m_0) + \sum_{i=1}^{k-1} \min_{u_i \in V_i}(\mu(u_i)r(m_i) + d(u_0,u_i)r(m_0m_i)) \tag{3.6}$$

The final step is to take the minimum among these values:

$$f_{min}(R) = \min_{u_0 \in V_0} f_{min}(R,u_0)$$

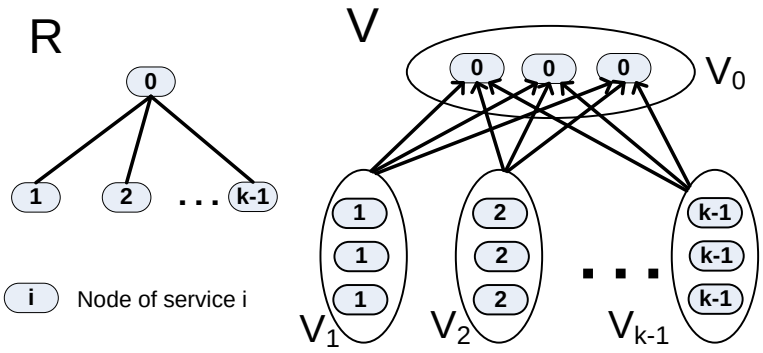


Figure 3.4: Calculating the minimal mapping cost with star graph request

Algorithm 2 Complex service mapping for Star graph

Input substrate graph G = (V,E,S), unity cost μ, abstract graph R = $(V^R, E^R, S), V^R = \{m_0, m_1, ..., m_{k-1}\}, E^R = \{m_0m_1, m_0m_2, ..., m_0, m_{k-1}\}$.

Output minimal mapping cost $f_{min}(R)$

1: V_i = all substrate nodes of service $s(m_i)$ for i=0,1,..,k-1.
2: Calculate $d(u_0, u_i) \quad \forall u_0 \in V_0 \; \forall i = 1..k-1 \; \forall u_i \in V_i$.
3: For all $u_0 \in V_0$ do
$f_{min}(R, u_0) = \mu(u_0)r(m_0) + \sum_{i=1}^{k-1} \min_{u_i \in V_i}(\mu(u_i)r(m_i) + d(u_0, u_i)r(m_0m_i))$.
4: $f_{min}(R) = \min_{u_0 \in V_0} f_{min}(R, u_0)$.

REMARK: If we denote by R_i the subgraph of R restricted in $\{m_i\}$, and $f_{min}(R_i, u_i)$ the minimal mapping cost as defined in section 3.5 for any $u_i \in V_i$, $i = 1, 2, .., k-1$; then $f_{min}(R_i, u_i) = \mu(u_i)r(m_i)$. Moreover, (3.6) can be rewritten under the recurrence form as follow: For any $u_0 \in V_0$,

$$f_{min}(R, u_0) = \mu(u_0)r(m_0) + \sum_{i=1}^{k-1} \min_{u_i \in V_i}(d(u_0, u_i)r(m_0m_i) + f_{min}(R_i, u_i)) \tag{3.7}$$

(3.7) seems to have no use in star graph request. However, it will prove useful in the following section.

3.7 The tree complex service

In this section, we extend the previous results form a path towards a tree topology complex service. Without losing generality, we consider m_0 as the root of R. Let R_i be the sub-tree of R rooted at m_i, i = 0,1,2,..,k-1. As before, we denote by $f_{min}(R_i, u_i)$ the minimal mapping cost of the subtree R_i if u_i is chosen as the substrate node of m_i. So $R_0 = R$ and if m_i is a leaf node of R, then R_i = singleton $\{m_i\}$ and $f_{min}(R_i, u_i) = \mu(u_i)r(m_i) \quad \forall u_i \in V_i$.

For any subtree R_i, we can consider the node m_i and its children as a star graph in which Eq. 3.7 holds true. Precisely,

Proposition 3.2. *For all i =0,1,2..k-1, and for all $u_i \in V_i$,*

$$f_{min}(R_i, u_i) = \mu(u_i)r(m_i) + \sum_{m_j child\ of m_i} \min_{u_j \in V_j} \left(d(u_i, u_j)r(m_im_j) + f_{min}(R_j, u_j)\right) \tag{3.8}$$

Proof. Consider the optimal map Φ_i of R_i should u_i is selected (which generates the cost

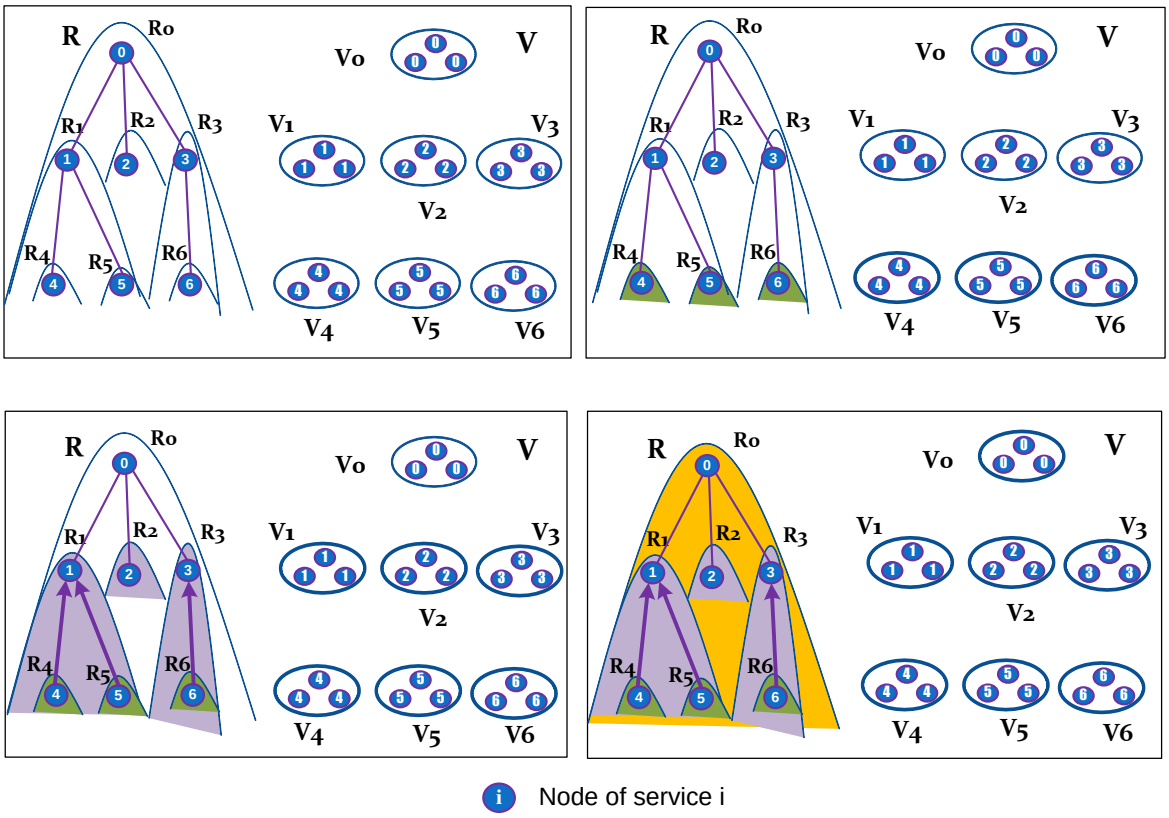


Figure 3.5: Alg. 3 execution: (a) Construct $V_0, V_1, V_2, V_3, V_4, V_5, V_6$ correspondent to each node of tree R. (b) The nodes of services 4,5,6 in R are the first to be calculated. (c) The nodes of services 1,2,3 are calculated in turn. (d) Node of service 0 is calculated, terminating the algorithm.

$f_{min}(R_i, u_i)$). Suppose that every child m_j of m_i is mapped to a substrate node $u'_j \in V_j$. By the definition of f_min, the mapping cost of R_j by Φ_i is at least $f_{min}(R_j, u'_j)$. Including the cost of edges $m_i m_j$, the mapping cost of Φ_i is at least:

$$f_{min}(R_i, u_i) = f(\Phi_i) \geq \mu(u_i) r(m_i) + \sum_{m_j \text{ child of } m_i} \left[d(u_i, u'_j) r(m_i m_j) + f_{min}(R_j, u'_j) \right]$$

$$\mu(u_i) r(m_i) + \sum_{m_j \text{child of} m_i} \min_{u_j \in V_j} \left(d(u_i, u_j) r(m_i m_j) + f_{min}(R_j, u_j) \right) \quad (3.9)$$

which is the right expression of (3.8).

Inversely, for every m_j child of m_i, let $u^*_j \in V_j$ be the substrate node which achieves the minimum

$$\min_{u_j \in V_j} \left[d(u_i, u_j) r(m_i m_j) + f_{min}(R_j, u_j) \right]$$

in Eq. 3.8. Let Φ^*_j be the map of R_j which maps m_j to u^*_j and generates the cost $f_{min}(R_j, u^*_j)$.

Algorithm 3 Tree Recursive Mapping (TRM) algorithm

Input substrate graph G = (V,E,S), unity cost μ, abstract tree R = (V^R, E^R, S), $V^R = \{m_0, m_1, .., m_{k-1}\}$, rooted at m_0.

Output minimal mapping cost $f_{min}(R)$

1: V_i = all substrate nodes of service $s^R(m_i)$ for i=0,1,..,k-1.
2: Calculate $d(u_i,u_j) \quad \forall i,j = 0,1..k-1 \; \forall u_i,u_j \in V_i,V_j$.
3: Create L the list of all nodes of R in decreasing order of their depth.
4: For any node in L from leftmost, until m_0 is visited:
5: m_i is the current node.
6: If m_i has children, then $\forall u_i \in V_i$,
$f_{min}(R_i,u_i) = \mu(u_i)r(m_i) + \sum_{m_j \text{child of } m_i} \min_{u_j \in V_j}[d(u_i,u_j)r(m_i m_j) + f_{min}(R_j,u_j)]$
7: else: $f_{min}(R_i,u_i) = \mu(u_i)r(m_i) \quad \forall u_i \in V_i$
8: Visit the next node
9: $f_{min}(R) = \min_{u_0 \in V_0} f_{min}(R_0,u_0)$.

If we define the map Φ_i^* as the union of all these Φ_j^* and $m_i \mapsto u_i$, then Φ_i^* will generate a cost expressed in the right side of (3.8):

$$\begin{aligned} f(\Phi_i^*) =& \mu(u_i)r(m_i) + \sum_{m_j \text{ child of } m_i} \left[d(u_i,u_j^*)r(m_i m_j) + f(\Phi_j^*)\right] \\ =& \mu(u_i)r(m_i) + \sum_{m_j \text{ child of } m_i} min_{u_j \in V_j}\left[d(u_i,u_j)r(m_i m_j) + f_{min}(R_j,u_j)\right] \end{aligned} \tag{3.10}$$

From (3.9) and (3.10), we have:

$$f_{min}(R_i,u_i) \geq f(\Phi_i^*)$$

By definition of f_{min}, we have: $f_{min}(R_i,u_i) = f(\Phi_i^*)$ which is also equal to the right side of (3.8).

□

This equation allows us to introduce the Tree Recursive Mapping (TRM) algorithm that is similar to the Algorithm 1: Starting from the subtrees corresponding to abstract leaf nodes, we calculate the minimal cost should a substrate node is chosen. Then, the cost for any subtree is calculated from its direct subtree descendants (i.e. R_i is calculated from all the R_j , m_j is a child of m_i) (Algorithm 3).

By sorting the nodes by the decreasing order of their depth (i.e. their distance to the root counted in hops - Node sorting can be done using reversed Breath First Search), each node

is listed after its children (Step 3). Thus the Step 6 is always validated. The execution of the Algorithm 3 for a requested tree of 7 nodes is illustrated in Fig. 3.5.

In this algorithm, there are two major steps that dominate the running time: the calculation of minimal distances at Step 2 and the loop at Step 4. While the distance calculation is the same as in Algorithm 1, the loop at Step 4 will take a number of operations equivalent to $\sum_{m_i m_j \in E^R} V_i V_j \leq E^R V^2$. So, if the distances can be calculated beforehand then the algorithm runs in time $O(E^R V^2)$ (or equivalent to $O(kV^2)$ - k is the number of nodes in the tree). If the basic services required in the complex service graph are distinguished, then the left expression in the inequality above is equivalent to $O(V^2)$.

The TRM algorithm is the final product of Section 3.5, 3.6 and 3.7, which is capable of finding the optimal map for complex service request in tree topology, including the two previous sections. However, attempting to extend TRM to cover general graph topology is futile. Indeed, general graph topology requests admit no algorithm that can find the optimal mapping cost in polynomial time. That is the result of the following section. We also present a heuristic solution for this case, based on TRM algorithm.

3.8 The general graph complex service

In the three previous sections, we have shown how optimal mappings for complex services with the path, star and tree topologies can be obtained in polynomial time. These algorithms are based on a recursive method: starting from the bottom level (leaves) to the top level (root), nodes of each level send the catalogue of their best choices to their direct higher levelled node (parent). The parent uses this information to calculate the best choice for all of its descendants (itself included). The fact that each node in the service graph has only one direct connection to higher or equal levels (its direct parent) makes service nodes a stand-alone decision maker for all of its descendants.

However, in service of general graph, a similar process is unable to obtain since there is no such a hierarchy. The circles in a graph make sure that any effort to classify the nodes in hierarchical levels always results in a node with more than one connection (adjacent nodes) to the higher or equal levels. Thus any decision passed down to this node must be in agreement of more than one node, preventing the recursive process.

In fact, mapping service graph in this case is not even polynomial time unless $NP = P$. The hardness of the problem is stated in the following proposition.

Proposition 3.3. *The Complex service graph mapping problem is a NP-hard problem*

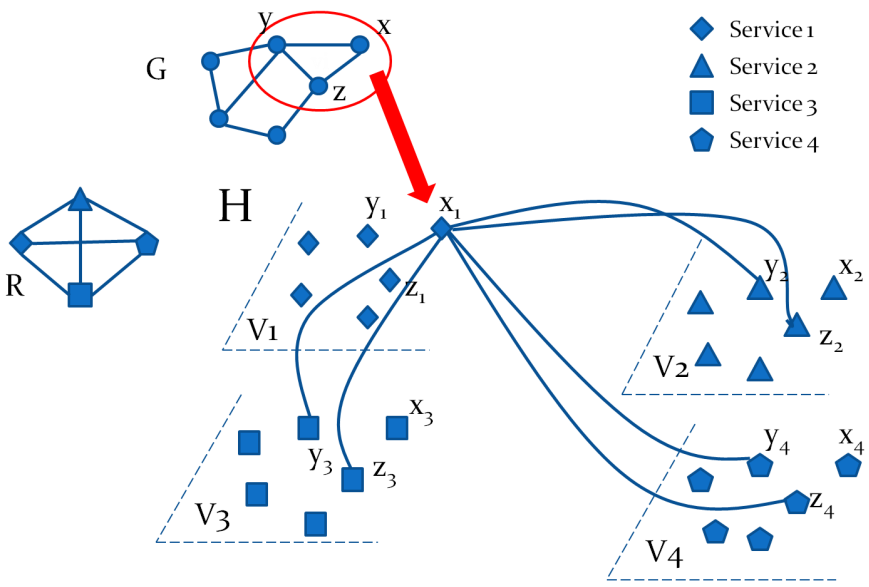


Figure 3.6: An example of constructing H. In this example, x has 2 adjacent nodes y,z in G. In the graph H, each duplicate of x is connected to all duplicates of y and z in other layer. The figure features only the neighbourhood of x_1 _ duplicate of x in layer 1

Proof. We will prove the NP-hardness of the problem by reduction from the CLIQUE decision problem. The CLIQUE problem is the decision problem in which given a graph $G = (V,E)$ and a number k, decides whether there is a clique (full subgraph) of size k in G. Precisely, from a given graph G and a number k, we will construct a request graph R and a substrate graph H and point out that whether or not there is a clique of size k in G is equivalent to whether or not the minimal mapping cost of R to H is $\frac{k(k+1)}{2}$. Consequently, we can say by certitude the existence of a clique of a given size in a given graph if we can solve the Complex service graph mapping problem.

Let R be the full graph of size k, $R = (V^R, E^R)$, where $V^R = \{m_1, m_2, ..., m_k\}$ and $E^R = \{m_i m_j, \forall 1 \leq i \neq j \leq k\}$. We associate m_i with a service i , $\forall 1 \leq i \leq k$, and construct a substrate graph $H = (V^H, E^H)$ such as:

- $V^H = V_1 \cup V_2 \cup ... \cup V_k$, V_i is a duplicate of V where all the nodes are associated with the service i. V_i is called layer i.
- E^H is constructed as follow: for every edge $xy \in E$, we link all the duplicates of x to the duplicates of y in different layers. Precisely, $E^H = \{x_i y_j \quad \forall 1 \leq i \neq j \leq k, \quad x_i \in V_i \quad y_j \in V_j$ such as $xy \in E\}$.

The construction of the graph H is illustrated by the Fig. 3.6.

We set the cost of every node and edge of substrate graph H to 1 per resource unit and every node and link resource demand of service request R to 1 unit. Since each node mapping

costs 1 and each edge mapping costs at least 1 (depending the length of the substrate link), the mapping cost of R into H of any map Φ is at least:

$$f(\Phi) \geq |V^R| + |E^R| = k + \frac{k(k-1)}{2} \tag{3.11}$$

$$=> f(\Phi) \geq \frac{k(k+1)}{2} \tag{3.12}$$

The equality happens in (3.12) if and only if each abstract edge of R is mapped into exactly one substrate edge of H (*).

If there is a clique L of size k in G, L = $\{l_1, l_2, ..., l_k\}$. We construct a map Φ' from R to H as follow:

- $\Phi'(m_i) = (l_i)_i, \quad (p)_i$ being the duplicate of p in layer i $\quad \forall p \in V$.
- $\Phi'(m_i m_j) = (l_i)_i (l_j)_j$.

This construction is valid since $l_i l_j \in E$, so by the construction of H, $(l_i)_i (l_j)_j \in E^H$.

Φ' maps each edge of E^R to exactly one edge of H, thus the cost of mapping E^R to H is $|E^R|$. We already know that the mapping cost of V^R to H is $|V^R|$. Therefore, the mapping cost of the map Φ' is:

$$f(\Phi') = |V^R| + |E^R| = \frac{k(k+1)}{2} \tag{3.13}$$

Adding (3.12) into consideration, Φ' is a minimal map from R to H, and the minimal mapping cost from R to H is $k(k+1)/2$.

Inversely, if the minimal mapping cost from R to H is $k(k+1)/2$. Let Φ^* be a minimal map from R to H. Since the equality in (3.3) happens, every abstract edge of R is mapped into exactly one substrate edge of H, i.e.:

$$\Phi^*(m_i)\Phi^*(m_j) \in E^H \quad \forall m_i m_j \in E^R$$

For all $p \in V^H$, we denote by $p|_G$ the projection of p in V, i.e. $p|_G \in V$ such that p is a duplicate of $p|_G$. Let C be the projection of $\Phi^*(V^R)$ in V, i.e. $C = \{\Phi^*(m)|_G, \quad \forall m \in V^R\}$

By the construction of H, there is no edge between any two duplicates of a same node in V. Therefore $\forall 1 \leq i \neq j \leq k$, $\Phi^*(m_i)$ and $\Phi^*(m_j)$ are not duplicates of a same node in V, i.e. $\forall 1 \leq i \neq j \leq k, \Phi^*(m_i)|_G \neq \Phi^*(m_j)|_G$. Hence C has exactly k substrate nodes.

Also by the construction of H, $\Phi^*(m_i)\Phi^*(m_j) \in E^H$ implies that $\Phi^*(m_i)|_G \Phi^*(m_j)|_G \in E$, which implies that C is a clique.

Therefore, C is a clique of size k.

The above arguments affirm that the graph G has a clique of size k if and only if the

Algorithm 4 Incremental graph mapping (IGM)

Input substrate graph G = (V,E,S), unity cost μ, abstract graph R = (V^R, E^R, S).
Output minimal mapping cost $f_{min}(R)$

Find $T = (V^R, E^T)$ the maximum spanning tree of R
$\Phi = TRM(G,T)$ the minimal map of T to G
Sorting $C^R = E^R \setminus E^T$ the remaining edges of R in descendant order.
While $C^R \neq \emptyset$ do:
    $mn = C^R(1)$ the first element in C^R
    $E^T = E^T \cup mn$
    Find $u, v \in V$ such that:
        $s(u) = s(m)$; $s(v) = s(n)$ and
        $\Psi : T \longmapsto G$ such that:
            - $\Psi(p) = \Phi(p) \quad \forall p \in V^R \setminus \{m,n\}$;
            - $\Psi(m) = u$;
            - $\Psi(n) = v$;
        has minimal mapping cost.
    $\Phi = \Psi$
    $T = T \cup \{mn\}$
    $C^R = C^R \setminus \{mn\}$
End While
$f_{min}(R) = f(\Phi)$

minimal mapping cost from R into G is $k(k+1)/2$, which is our objective.

Thus the resolution of CLIQUE cannot be harder than the Complex service mapping problem. □

The conclusion of Proposition 3.3 prevents any attempt to find an exact solution of the complex service mapping in polynomial time. A naive approach will be to directly apply the recursive tree algorithm to a spanning tree of the request graph. However, this approach could only use a subset of abstract edges (equal to $|V|-1$ elements) over the overall existing ones. It may work well for sparse graph (low average degree); however, with a dense graph, the ratio between the total weight of all abstract edges of the request graph ($|E|$ edges) and one of the spanning tree ($|V|-1$ edges), even the maximum spanning tree, may reach $|V|/2$ in the worst case (all edges are equally weighted), which significantly increases the odd between the result and the optimal mapping cost.

For this reason, we propose to reinforce the tree recursive mapping by successively mapping remaining edges one by one. First we apply the tree algorithm to the maximum spanning tree of the request graph. The maximum spanning tree mapping ensures that all ab-

stract nodes and the largest portion of abstract edges that cover the entire abstract nodes are mapped into the substrate network graph with the minimal cost. Then, we repeatedly map the remaining abstract edge, starting from the heaviest edge (having the highest requested resources) to the least weighted edge. Mapping each abstract edge *mn* consists of finding the target nodes of *m* and *n* that minimize the mapping cost, while the mapping of other substrate nodes should remain static. This algorithm, which is called Incremental Graph Mapping (IGM), considers the overall edge mapping, and not solely the spanning tree, thus avoiding the limitation of TRM.

However, this does not mean that the minimal map for the on-processing part of request (T in Algorithm 4) is found at each iteration (otherwise we would obtain a polynomial algorithm for a NP-hard problem!). The resulting map from the loop IGM Step. 7 is a local optimal map that differs from the current map at two substrate nodes, while the global optimal map may require more changes from that same map. Adding a new heavy weighted edge may need more change from the current map to obtain the optimal map than a light weighted edge; therefore it is reasonable to start mapping edges from the heaviest one to the lightest one. For the same reason, it is necessary to map the maximum spanning tree, so that the largest set of covering edges are mapped first in an optimal manner.

Aside from the TRM complexity, the IGM algorithm necessitates finding for each remaining edge a pair of substrate nodes that minimizes the mapping cost. In the worst case, this consists of checking all pairs in V, i.e. the whole loop is bounded by $E^R V^2$. Therefore, the algorithm terminates in time $O(V^3 + E^R V^2)$.

3.9 General graph mapping evaluation

3.9.1 Experiment objectives and setting

3.9.1.1 Experiment objectives

In this section, we evaluate the performance of our algorithm through simulation. We will compare its performance with ViNEYard [38, 39]. We also study the evolution of the performance when the density of request graph increases. As previously discussed, TRM imposes the mapping of the maximum spanning tree over the whole request graph which may degrade its performance, if the overall requested resources is several times greater than that of its maximum spanning tree, such as in dense graphs. If all the edge weights are drawn from the same distribution law, then the ratio between the sum of weight of all edges of the request and sum of weight of all edges of the maximum spanning tree is proportional to

E/V. Therefore, the denser the request graph is, the higher this ratio becomes, which makes the TRM application less effective. The simulation aims to highlight how the enhancements in IGM overcome this limitation. For that we have implemented an algorithm that returns the map by directly applying TRM to the maximum spanning tree of the requests. This naive algorithm, which we call Spanning Tree Recursive Mapping (STRM) will also be evaluated in the simulation.

3.9.1.2 Setting

Parameters	Values
Number of nodes per substrate graph	100
Number of services per substrate graph	15
Node processing cost per unit	Uniform in [20,40]
Edge usage cost per unit	Uniform in [20,40]
Number of nodes per request graph	[5:30]
Tree node resource request (unit)	Uniform in [5,20]
Tree edge resource request (unit)	Uniform in [5,20]

Table 3.4: Experiment configuration

First of all, we generate substrate network graphs using Barabasi-Albert model [19]. Each graph consists of 100 nodes where each node provides one of 15 services drawn at random. The node and edge costs are drawn uniformly randomly in range [20-40]. We believe that in reality, when a new node is constructed, it has more interest to connect to a more important node than a less important one to improve the chance of taking requests. Thus the most important nodes are the ones who have the most connections. As the consequence, the higher degree a node has, the more chance it has to connect to the new node. This is the reason why we choose Barabasi-Albert model for the substrate graph.

If we call p the probability that an edge is taken into the request graph ($0 \leq p \leq 1$), then the average ratio between the total weight and the maximum spanning tree is calculated as follows:

$$\frac{E^R}{V^R} = \frac{p(V^R)^2/2}{V^R} = \frac{pV^R}{2}$$

p represents the density of the graph (*density indicator*) : the higher p is, the denser the graph is. In these simulations, we set four values of p, $p = 0.1, 0.4, 0.7, 1.0$. The graphs with $p = 0.1$ are sparse graphs, and graphs with $p = 1.0$ are cliques (full graphs).

We generate a set of random requests ranging from 5 to 30 nodes each, with one of four density indicator $p = 0.1, 0.4, 0.7, 1.0$. As we classify the request graphs in number of nodes and the density indicator, $Erdos - Renyi$ model [49] is the natural choice to generate

these graphs, since it can provide random graphs bases on number of nodes and edges. Each node and edge resource requirements are uniformly distributed in range [5-20]. The testbed details are listed in Table 3.4. The testbed is realised on a Dell Optiplex computer Windows XP with 2GHz Duo Core processor and 5Gb RAM. Due to the memory needed by ViNEYard, we couldn't increase the number of nodes in the request graphs further than 30 nodes.

3.9.2 Reference algorithm for comparison

Due to the fact that we cannot obtain the exact optimal map, we have chose ViNEYard algorithm [38, 39] to compare the performance with. ViNEYard is known to obtain better results compared to other mapping algorithms [38, 39]. The algorithm solves a multi-flow commodity Mix Integer Program by using Relaxation and Rounding technique. A brief description of the approach is described in Section 3.3. We have implemented this algorithm in our experiment relaxing some capacity constraints. This is to reduce greatly the number of equations to process but it still remains a complex problem of $O(E^R E)$ variables and $O(E + E^R V)$ equations. We use the performance of ViNEYard to evaluate that of STRM and IGM.

3.9.3 Evaluation method

We apply the IGM, STRM and ViNEYard algorithms to process the requests separately. Each algorithm is applied to the same set of substrate graphs, and process the same set of requests generated as explained previously. We measure the mapping cost of each request, the execution time by each algorithm. Since execution time depends on the implementation method, the result is considered as relative indicator and not absolute one. We trace the evolution of the results with the number of nodes per request service graph and the request graph density indicator p.

3.9.4 Evaluation results

3.9.4.1 Overall performance

Fig. 3.7 depicts the service mapping cost of each algorithm versus the number of nodes per request graph. In the figure, the values are average mapping cost taken from mapping all request graphs of the same number of nodes and the vertical bars indicate the correspon-

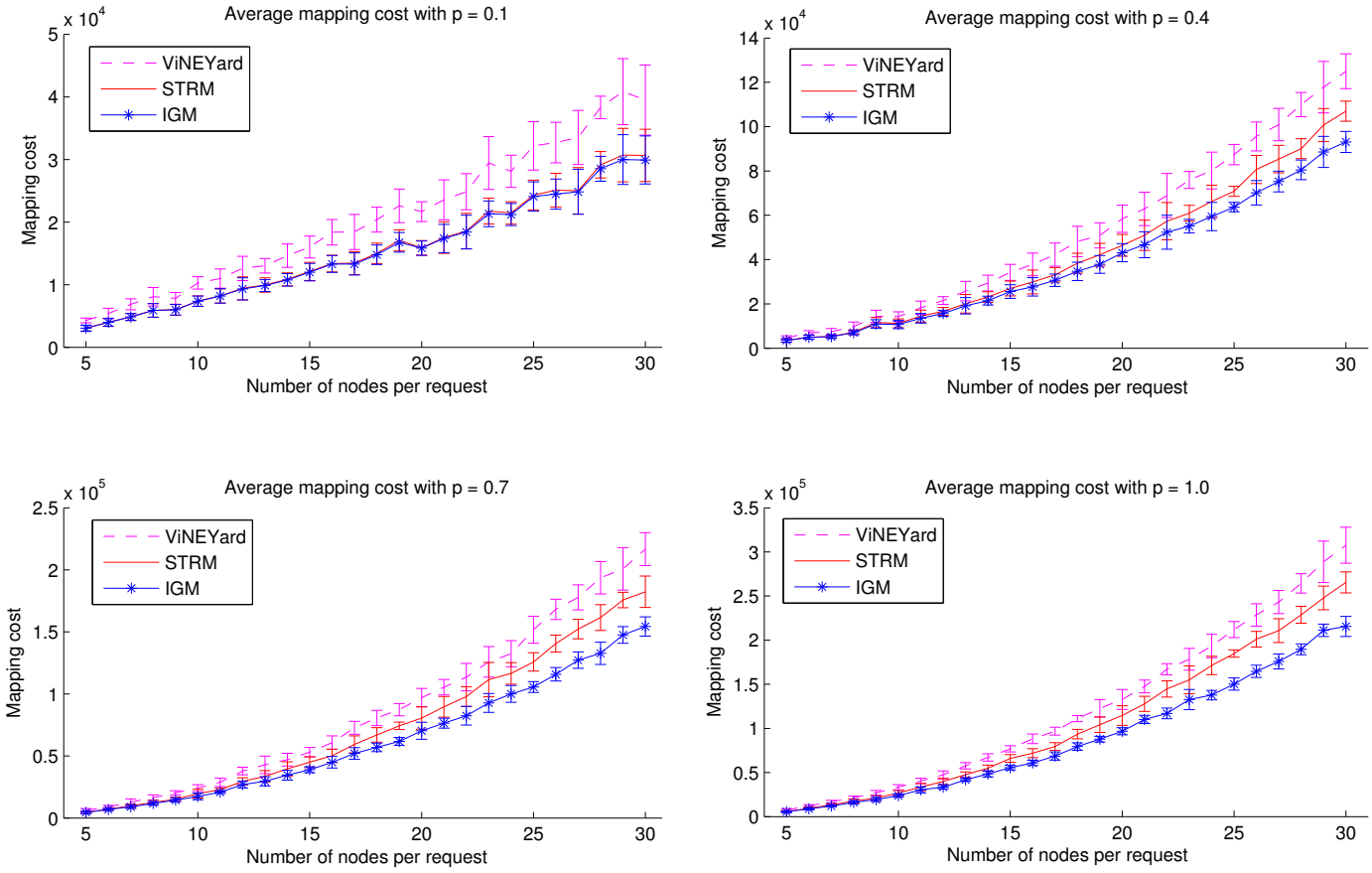


Figure 3.7: Mapping cost by number of nodes per request service graph

dent standard deviations. The figure shows that IGM performances the best performance, followed by STRM, and ViNEYard terminates the last regardless of the density of the request graph (p = 0.1, 0.4, 0.7, 1.0). The distance between the values from IGM and STRM increases with the number of nodes per request graph. This is explained by the fact that with bigger request graph (with the same density indicator p), the ratio between the total weight of the request and its maximum spanning tree increases, therefore STRM would likely produce a solution far from optimum. IGM, however, reconsiders the map of STRM by continue mapping the remaining abstract edges, which effectively adjusts the global map. The distance between IGM and STRM is the consequence of this enhancement, which proves to be important when request graphs get larger.

The second remark is that the higher p is, the more STRM is closer to ViNEYard and further from IGM. At p = 0.1, STRM and IGM are almost identical. For p = 0.4, 0.7 and 1.0, the distance between the two is significant for large requests. The explanation stays the same for this, as p increases the ratio between the total weight of a request and its maximum spanning tree.

While also considering all abstract edges in its execution, ViNEYARD suffers from its rounding. Solving the linear program results in fractional values associated with each substrate candidate nodes of each abstract node. Rounding technique helps choosing among all candidates a unique one that will be selected as the target of the abstract node. However, so far there is little relation between the associated values and the map performance. Therefore the more candidates and the number of requested edges are, the less likely that the target choice is a good one, be it deterministic choice (taking the maximal associated value) or probabilistic one (drawn randomly following the associated values). Unlike IGM which quickly tries re-mapping at each iteration, ViNEYard cannot reconsider the map since it necessitates resolving another linear program. All rounding decisions (target choices) are made in parallel for all abstract nodes without considering the fact that the choice of one abstract node may affect others'. Thus for dense service requests, its performance falls far behind IGM's.

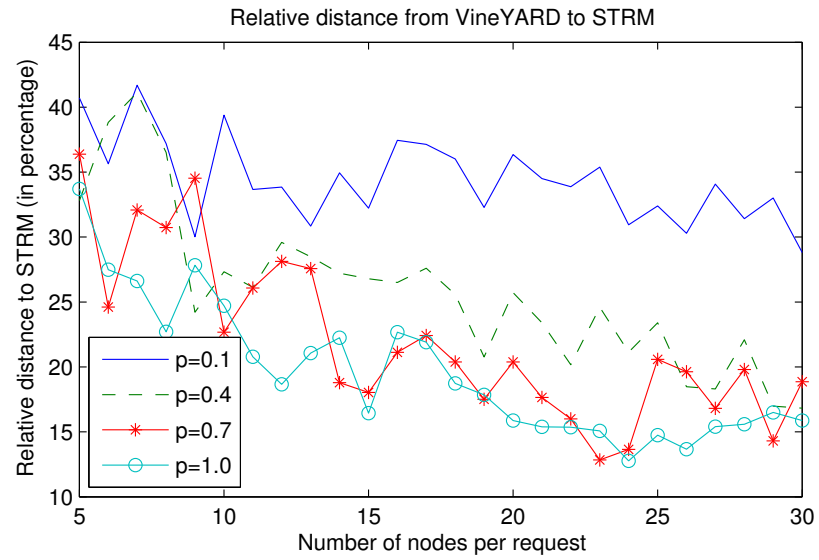
3.9.4.2 The dependence of request graph density

Fig. 3.8 shows the relative distance[2] from ViNEYard and IGM to STRM results. As predicted, the STRM algorithm becomes less effective as the request graph gets denser (higher p). For the same number of nodes, the relative distance between ViNEYard and STRM decreases in function of p. STRM starts with a good performance compared to ViNEYard; it obtains 30-40% less cost than ViNEYard at p = 0.1. However, as p increases, the ViNEYard's performance get close to STRM, down to roughly 15% higher than STRM at p = 1.0 and 25 nodes.

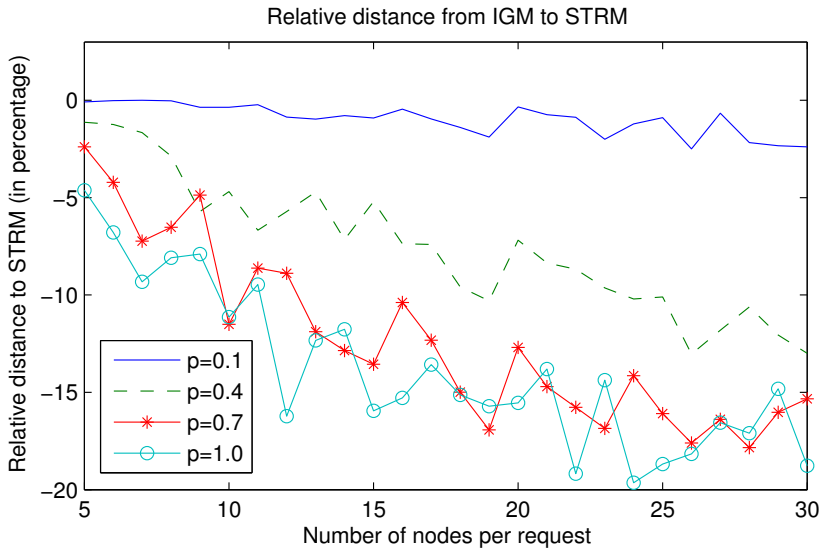
The second remark is that if we keep p constant, then the relative distance between the ViNEYard and STRM has the tendency to decrease as the number of nodes increases. This confirms our analysis that the performance of STRM is degrading as the number of nodes increases, illustrated by the decreasing distance curves. We believe that if we increases the number of nodes further, the performance of ViNEYard will catch up and exceed that of STRM. However, due to the memory consumption in solving the linear program of ViNEYard, we couldn't realize the testbed beyond 30 node per request graph.

Still, the performance of ViNEYard is not getting better than STRM until the request graphs are sufficiently large (number of nodes > 30 for p = 1). In real world where customers rarely require more than 30 nodes in a request, its performance is not high enough to be of any practical use.

[2]The relative distance between ViNEYard, IGM, and STRM are calculated following the formula $\frac{ViNEYard-STRM}{STRM} \times 100\%$ and $\frac{IGM-STRM}{STRM} \times 100\%$ respectively



(a) ViNEYard vs. STRM



(b) IGM vs. STRM

Figure 3.8: ViNEYard & IGM vs STRM

We observe the same trend between IGM and STRM. Their performances are close when the density indicator of the request graphs are small. As the number of nodes and the density of request graphs increase, so does the ratio between the total eight of request graph's edges and the total weight of its maximum spanning tree; the distance between IGM and STRM gets further and further, attending 18% at 30 nodes and p = 1.0. As argued previously, the lack of consideration of remaining edges makes the STRM's performance becomes further and further behind IGM as the number of nodes increases, resulting in the decreasing tendency of the curves as shown in Fig. 3.8(b).

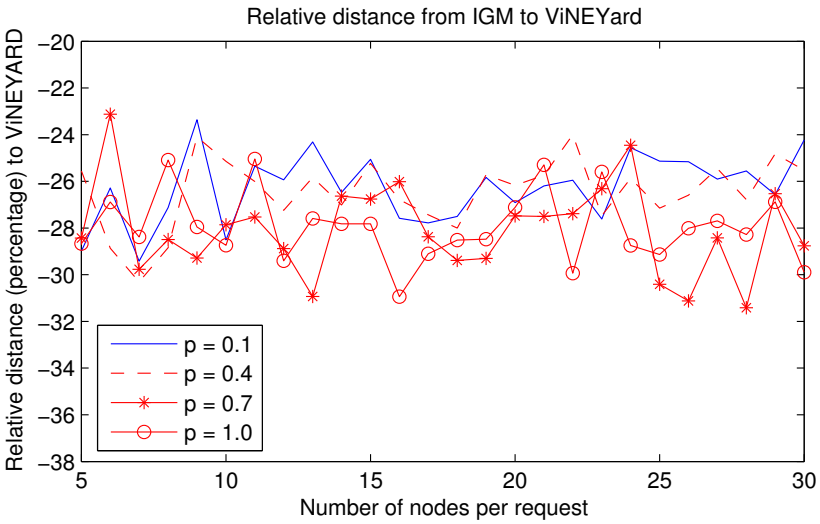


Figure 3.9: IGM vs. ViNEYard

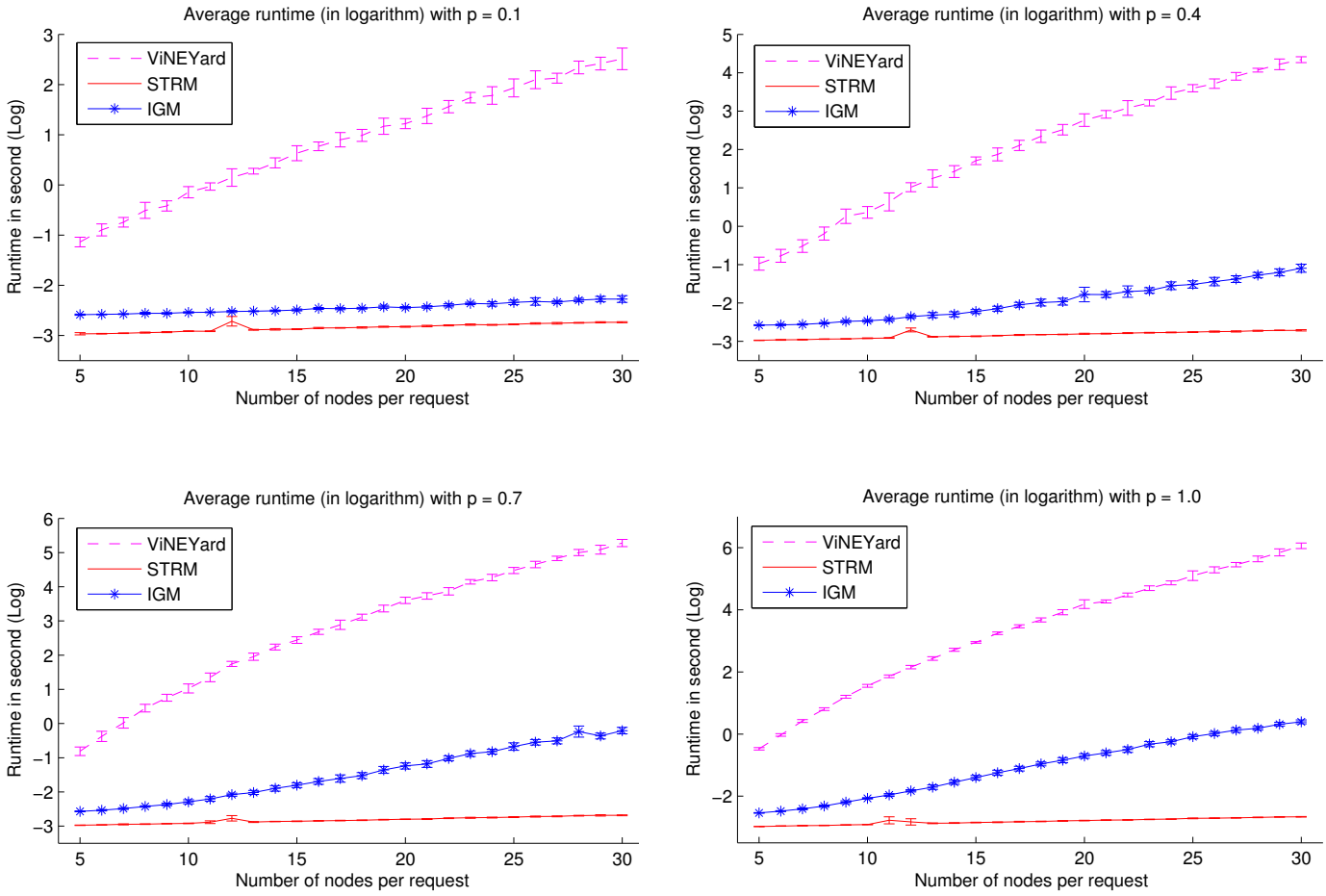


Figure 3.10: Algorithm execution time by number of nodes per request service graph

Not only that IGM returns better solution than ViNEYard, its starts with a good stand, that of STRM, even for small p and number of nodes, while ViNEYard's starting performance is further down. Thus IGM can be applied to any request and still achieve good result regardless of its size. In fact, its solutions are between 22-32% less costly than that of ViNEYard (Fig. 3.9). Although we cannot make comparison above 30 nodes, we observe a rather stable ratio between ViNEYard and IGM in this figure, which claims the advantage of IGM compared to ViNEYard.

3.9.4.3 Execution time

Another advantage of IGM is its complexity: it requires much less time and memory than ViNEYard to run. With a computer of 2GHz Duo Core processor and 5Gb RAM, we can run IGM with a request up to 100 (or even more!) in less than 2 seconds. On the same computer, we cannot apply ViNEYard to a request of more than 30 nodes due to the lack of memory, and the algorithm takes nearly 500 second to finish (Fig. 3.10). ViNEYard runtime is long due to its large linear programme which consists of $O(E^R E)$ variables and $O(E + E^R V)$ equations.

3.10 Conclusion

This chapter introduced three algorithms for solving the complex service mapping problem in cloud infrastructure for different topologies: path, star, tree and general graph. Although the latest proposed algorithm covers all topologies. Our algorithms complexity is $O(V^3) + O(E^R V^2)$; however, if the distances between all pairs of nodes are calculated beforehand, the algorithms' complexity is $O(E^R V^2)$, or $O(V^2)$ given the fact that the required services are distinguished. The experiments show that the algorithms not only give good results in term of mapping cost but also in terms of execution time. We have considered that the underlying network resources are unlimited, which is reasonable in the context of cloud computing.

Chapter 4

Adaptive cost-optimized service placement

4.1 Introduction

In the previous chapter, we address the problem of service composition where a requested service is dynamically composed of basic services that are deployed in the network. This approach helps the service provider to make the best choice when deploying customers' services. The solution, however, does not consider the demand of end-users who access to the requested service. In this chapter, we address the problem taking the end-users' demands into consideration. With so many demands of end-users from various locations in the network, one service instance alone will not be sufficient. We propose to deploy not one, but a set of service replicas (copies) in the network. This chapter will present our solution to find the best strategy to place these copies in the network.

We consider the same Online Video Streaming scenario presented in the previous chapter (Table. 3.1). In the previous chapter, our cloud customer, the media company, has contracted a service provider to deploy a Online Video Streaming service targeting the French market. After the business success in France, it now is willing to expand its market to all Europe. It will contract the service provider again for a solution that assures the quality of service to its clients (end-users) with the minimal cost, as usual. Should the service provider deploys an Online Video Streaming service similar to the one deployed in France in every country in Europe? The answer is obviously not. Building such a complex service is costly, it may not be profitable for small markets such as Luxembourg. A better strategy for the service provider is to use the French service for neighbouring countries such as Luxembourg and Lichtenstein. This, however, does not come free of charge. Assuring the quality of service

delivered to end-users far from the service's location necessitates additional network and system resources (bandwidth, for instances) that costs money to the service provider and the media company. Therefore, the service provider has to consider all possibilities of locations for replicas hosting (copies of its service) and evaluate the cost of these choices, including the cost of installing the service replicas (*service installation cost*) and the cost of delivering the services to the end-users (*service delivery cost*).

Another issue is that the users' demands are constantly changing. Even if the service provider finds best locations for current demands, it may not remain the best choice for future. Fortunately, the Cloud Computing is designed for that purpose. Cloud Computing allows to quickly deploy/terminate, scale up/down services as requested and pay for what is actually used. Changing service component locations is easy enough within the cloud infrastructure. Therefore the company does not need to provision far ahead for the future. They can make the provisioning plan for current period, then change it later according to the demands of its clients. However, too many modifications to the provisioning plan in a short period of time may lead to undesired cost. Firstly, cloud infrastructure providers whom the service provider draws the resources from may charge outbound traffic from their datacenters. Thus sending data and content from a datacenter to a new one to establish a replica may add an extra cost to the service provider. Secondly, due to the limited network bandwidth, it may take a lot of time to finalize the transfer of large data and content on Internet, as discussed in 2.8.4. Other issues have to be treated as well, such as service discovery, service requests redirection, etc. Therefore, modifying the provisioning plan too frequently would disrupt the service, and may violate the SLAs. Hence, even if modification is needed, the service provider has to consider it with care. They certainly don't want to remove all the service replicas and deploy whole new ones just for a marginal benefit.

Problem statement

The above scenario illustrating the service placement problem in the cloud environment can be formulated as follows:

Given a set of locations which are available for services' set up. Setting a service at each location raise a *service installation cost* depending on the particular location. Each service open up in a location can be used to serve not only clients of the same location but also clients from other locations with certain *service delivery cost* depending on the distance from the clients' sites and the service's location as well as the level of their demands. The service placement problem respond to these following questions:

- How to choose a set of locations (provisioning plan) to set up the services that mini-

mizes the total cost, including the installation cost and the service delivery cost?
- In case of change in the demands, how to modify the provisioning plan to minimize the total cost while keeping the modification as small as possible?

The first question can be seen as a use case of Facility Location Problem (FLP), particularly under the metric distance. Given a network graph where each node is either a client or a location where a facility may be built/opened, the FLP tries to find a set of locations to set up facilities in order to minimize the total cost. Traditionally, FLP deals with the problem of placing large facilities such as storages, supermarkets, or distribution centers which once installed, could not or should not be removed. In reality, the owners of the facilities only want a long term solution, with little modifications in the future. Such facilities are much more costly compared to the cost that is needed to serve clients. Thus replacing one facility by another is only considered in extreme condition.

In contrary, service placing strategies in the cloud have to deal with the ever changing environment. While traditional FLP refers to large costly facilities, cloud services can be easily deployed and removed. In cloud environment, a service can be as simple as a binary image that providers can upload/remove to/from a local server. Replacing a service replica by another is feasible using virtual machine migration process, even with its local persistent state and network connections [23,30]. Thus service provider can reconfigure their services' locations to fit with end-users' changing demands. Still, as mentioned, modifications to the provisioning plan should take into consideration the trade-off between the benefit and the cost such a modification may bring back. That is the meaning of the second question that we need to respond to.

In this chapter we will propose a scheme to find the most suitable modifications to the current provisioning plan based on Facility Location model. Given clients' demand, we aims to find a new provisioning plan that minimizes the total cost, but also requiring as few modifications as possible. We make the assumption that a mechanism to redirect clients' requests to suitable replicas already exists. Many existing mechanisms can assure this task, such as dynamic DNS, service discovery ([4,8,10,129]), distribution based on group structure [132] or direct request routing [107]. We also assume that there is no limited capacity in this problem (Uncapacitated Facility Location).

4.2 Related works: Facility Location

As discussed above, FLP treats the problems of minimizing the total cost of a heavy structure deployment which consists of the installation cost and service delivery cost. Since

FLP is a NP-hard [68], these works aim to find approximation algorithms to the optimal static deployment.

Lin and Vitter [82] present a rounding method for linear program relaxation to find approximately minimal number of locations under the condition that the sum of distances to clients are bounded by a given number. Based on this method, Shmoys, Tardos and Aardal [120] propose an algorithm with an approximation factor of $3/(1-e^{-3}) \approx 3.16$. [66] combines the algorithm of Shmoys, Tardos and Aardal with greedy heuristic to achieve a factor of 2.408. Moreover, they prove that any rounding strategy applied to a linear programming method cannot exceed the bound of 1.463-approximation factor unless $NP \subset DTIME[n^{O(loglogn)}]$. Mahdian [83] proposes a 1.861-approximation greedy algorithm. Later he and Jain [73] present a greedy algorithm with an approximation factor of 1.61. Then he combines their idea with cost scaling to obtain an approximation factor of 1.52 [84]. Korupolu et al [77] propose a local search method to find a 5-approximation solution. Pandit and Pemmaraju [105] are more interested in the trade-off of runtime used resources (communication rounds, message size, etc) in a network and the quality of solution. Using primal-dual method, they obtain a 7-approximation algorithm running in O(log m + log n) rounds with m facilities and n clients, and an approximation factor of $O(m^{2/\sqrt{k}}.n^{3/\sqrt{k}})$ for a k-round algorithm. Zhang et al [136] propose a dynamic placement method that is 27-approximated in worse case, but simulations suggest a close to greedy method in normal use case.

These algorithms try to find the best provisioning plan that minimizes the cost of deploying service replicas. However, they don't take into account the changes in the network, nor required modifications for the current provisioning plan [136]. The purpose of our scheme is to reinforce these algorithms to cope with the changes in the network. We will introduce a new formulation of the problem to calculate a new provisioning plan taking into consideration the current service replicas installation. To the best of our knowledge, it is the first work to ever consider this aspect. The advantage of our scheme is that the formulation is in the same form as the facility location problem, and therefore we could make use of existing contributions.

4.3 Service replica placement optimization

Let's consider a connected network graph $G = (V,E)$ in which each node is a potential host for a service. Each node $p \in V$ has a certain level of users' requests for service w_p, and is associated with a cost for installing & maintaining the service (*installation cost*) c_p (w_p and c_p are defined *per time unit*). For any node that does not have an infrastructure to install

the service, $c_p = +\infty$. Each edge ab of the graph has a cost e_{ab} for delivering a service quantity unit. We denote by S the *provisioning plan* (or *service set*), i.e. set of nodes where a replica of service is installed.

We categorize the cost into two types: *installation cost* and *delivery cost*. The installation cost c_p represents the cost of installing and maintaining a service replica at a node p. For a service set S, the installation cost c^S is:

$$c^S = \sum_{i \in S} c_i$$

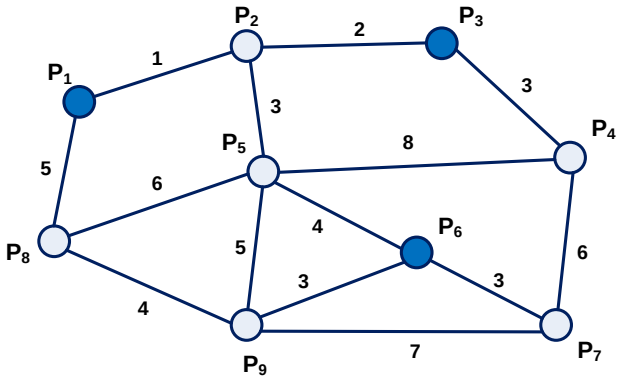


Figure 4.1: $S = \{P_1, P_3, P_6\}$; $u^S_{P_7}$ depends on distance from P_7 to S: $d(P_7, S) = d(P_7, P_6) = 3$

Another cost to consider is the delivery cost u^S which represents the impact of the distance between the service replicas and the clients in terms of network resources. It is a distance-related cost that increases with the distance between the service replicas and clients. Demands from nodes far from service set require more efforts to satisfy than demands from nodes nearer to service set. For every node $p \in V$, the delivery cost of p determines the effort required from service provider to assure the quality the service for this node. It is calculated by the product of the quantity of service requests and the distance from this node to the service set.

$$u^S_p = w_p * d(p, S)$$

in which d(p,S) denotes the shortest distance from the node to a service node.

The total delivery cost u^S is:

$$u^S = \sum_{p \in V} u^S_p$$

Let's consider the previously presented Online Video Streaming service example (Chapter

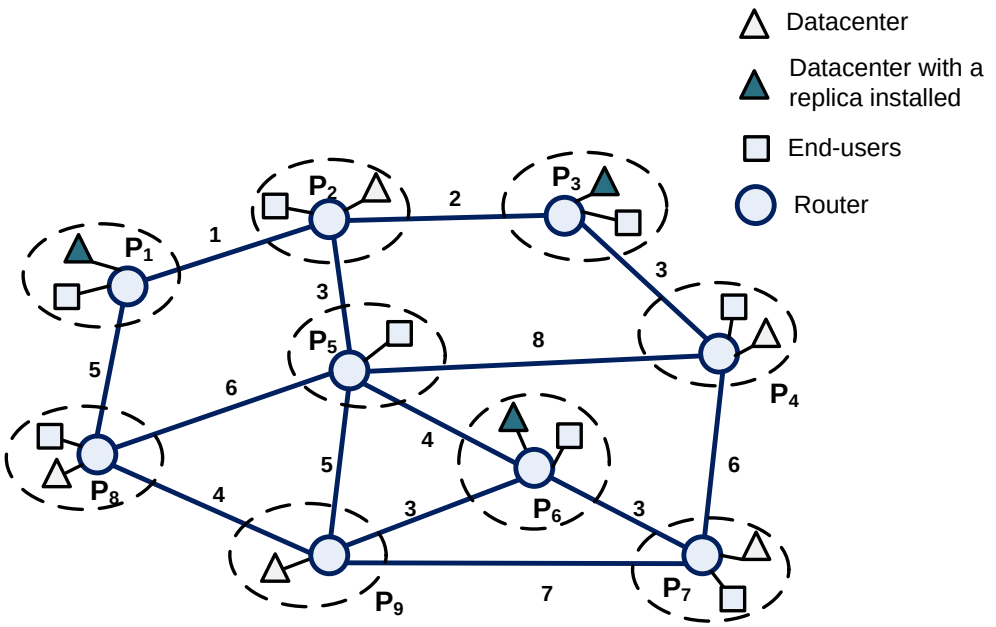


Figure 4.2: Online Video Streaming example. Each vertex P_i is an autonomous system that containing clients and/or datacenters.

3). Suppose that the network consists of many autonomous systems (AS). Each AS contains routers, client access points and/or datacenters ready to install the service replicas (Fig. 4.1). We model the problem by representing each AS by a vertex P_i which quantity of service request is summed up from all requests from the clients inside the AS. The installation cost and service delivery cost are determined as follows:

1) At each vertex P_i, the installation cost is calculated using IGM algorithm (Chapter 3) by adding "P_i" as the value of the "location" requirement of the Transcode node. Since all media flow will be streamed directly from Transcode node to clients' terminals, we can ignore the other two components (Storage and Portal) and consider the complex service simply as a Transcode node. In this chapter, the installation cost is considered as input for the service placing problem. If there is no infrastructure that is capable of setting up the service (i.e. there is no datacenter or datacenters do not meet the service's requirements), then the installation cost is considered as infinity $+\infty$ (e.g. P_5).

2) To satisfy the requests of clients in a particular AS, the service provider has to provision sufficient resources from one of its copies to this AS. The delivery cost is what the service provider (and eventually cloud customer) has to pay to the network operator for the link usage. For example, in the Fig.4.2, suppose that replica P_6 is the nearest to P_7 which has the quantity of request w_7 equivalent to 50,000 GB during a period of 1,000h, which is equivalent to an average demand of 13.9 MBps and the 'distance' $d(P_7, P_6)$ is the

cost of allocating the link bandwidth between P_7 and P_6 : $d(P_7,P_6) = 3USD/MBps/h$. The cost to reserve the link resource for the service request from P_7 for this period is $u^S_{P_7} = w_7 * d(P_7,P_6) = 41.7$ (USD per hour).

The total cost for a service set S is:

$$\begin{aligned} TC^S &= c^S + u^S \\ &= \sum_{i \in S} c_i + \sum_{p \in V} d(p,S) * w_p \end{aligned} \tag{4.1}$$

There is a special case where $S = \emptyset$, i.e. no service is deployed in the network. Since it is a particular case, we can ignore it and suppose that $S \neq \emptyset$, which justifies the distance d(p,S) in (4.1).

The *service placement problem* is the problem that given a network graph, how to find a provisioning plan that minimizes the total cost.

We denote by d_{pi} the shortest path from node p to node i in G. Let $x_i \in \{0,1\}$ be the variable that indicates whether the node i is selected into the provisioning plan ($x_i = 1$) or not (x_i=0). Let $y_{pi} \in \{0,1\}$ be the variable that indicates that whether the clients at node p is served by the service at node i (y_{pi}=1) or not (y_{pi}=0). The service placement problem can be written in integer linear program as follow:

$$\text{Min} \quad \textstyle\sum_{i \in V} c_i x_i + \sum_{i,p \in V} d_{pi} w_p y_{pi} \tag{4.2}$$

$$s.t. \quad y_{pi} \leq x_i \qquad \forall i, p \in V \tag{4.3}$$

$$\textstyle\sum_{i \in V} y_{pi} \geq 1 \qquad \forall p \in V \tag{4.4}$$

$$x_i, y_{pi} \in \{0,1\} \tag{4.5}$$

The Eq.(4.3) signifies that in order for clients at p to be served by service at node i, i has to be selected into the provisioning plan. Eq. (4.4) indicates that every client is served (i.e. at least one of $y_{pi} = 1$). The objective of the problem is to find a service set S that minimizes the total cost TC^S. However, this is a difficult task. The hardness of the problem will be provided in the following section.

4.4 Hardness of the service placement problem

Proposition 4.1. *Service placement problem is NP-hard*

We will prove the NP-hardness of the Service placement problem by the reduction from

the SMALLEST DOMINATING SET problem [1]. Given a graph G = (V,E), we set the installation cost to 1.5 for every node and edge cost to 1 for every edge. Now set the quantity of end-users' demand at every node to 1. We will prove that the minimal service set is also the minimal dominating set.

Proof. Suppose that S is the minimal service set. Suppose that there is a node a such as $d(q,S) > 1$. Since the length of every edge is integer, so $d(q,S) \geq 2$. Let S' = $S \cup q$. We have:

$$\begin{aligned} & \forall p \in V \backslash q, \quad d(p,S) \geq d(p,S') \\ \Rightarrow \quad & \forall p \in V \backslash q, \quad u_p^S \geq u_p^{S'} \end{aligned} \tag{4.6}$$

Moreover, $$u_q^S = d(a,S) * w_q \geq 2 > c_q + u_q^{S'} \tag{4.7}$$

From (4.6) and (4.7), we obtain:

$$TC^S > TC^{S'}$$

which is contrary to the definition of S. Hence there does not exist $q \in V$ such as $d(q,S) > 1$. So $d(p,S) = 1 \quad \forall p \in V \backslash S$, which means that $\forall p \in V \backslash S$, p is joined to S by some edge, i.e. S is a dominating set.

For any dominating set D,

$$\begin{aligned} TC^D &= \sum_{v \in D} c_v + \sum_{v \notin D} d(v,D) * w_v \\ &= 1.5 \times |D| + 1 \times (|V| - |D|) \\ &= |V| + 0.5|D| \end{aligned}$$

Therefore, let M be the minimal dominating set, we have $TC^M \leq TC^S$, the equation happens if and only if $|M| = |S|$. By deduction from the definition of S, S is also a minimal dominating set. □

The above proof shows that the service placement is also NP-hard. FLP proposes many

[1] SMALLEST DOMINATING SET: given a graph G, determine the smallest dominating set, i.e. a set of vertices such as all remaining vertices are adjacent to one of its vertices.

integer program algorithms to find approximation solution[2] for the system (4.2). They are also used in several greedy algorithms to find the approximation factor.

The linear programme approaches begin with the integer linear programme (4.2). Due to the fact that the problem is NP-hard, exact solution to this problem is not possible. Instead, authors relax the integer constraints (4.5), transforming the problem into a linear programme which can be solved in polynomial time. The downside of this, of course, is that the resulted variables are fractional, not integer. Thus they have to "round" these variables into integer values, which are taken as final solution. Depending on the rounding techniques, the approximation factor can be varied, with the best factor so far is 1.52 [84] as discussed in the section 4.2.

Although providing good approximation solutions, these algorithms only focus on how to get the minimal total cost given the current demand. They can be good choices for the start-up phase where the replicas are first installed in the network, which requires the forecasting data on service demand only. However, in the long term it is inefficient to handle the changes in the cloud such as changes of quantity level of requests or network cost. One possibility is to divide the time in periods, and for each period we execute these algorithms based on the knowledge of the current network topology and clients' request rate gathered by the replicas at each period. However, each execution of these algorithms may result in a very different service set, resulting in radical modification of replica deployment. Even dynamic solutions still lack the consideration of the modifications needed versus the benefit from these modifications. While service replicas can be easily deployed/replaced by allocating/removing resource in a virtualized environment like Cloud Computing, such a modification still needs time and effort (cost) for clients' requests to reach to the new service replicas and establish service delivery mechanisms, thus provoking disruptions of service and increasing associated costs (service discovery, service delivery, QoS negotiation, etc). Therefore, to adapt to the changes in the cloud, we need to take into account the number of replica modifications (i.e. the difference in service sets).

4.5 Adaptation to changes

Suppose that S^0 is the service set generated at the period t_0. At the period t_0+1, we want to find a new service set that reduces the TC while considering the modification made to S^0. If the current context is not very different from the one at t_0, then we prefer a new provisioning

[2]General non-metric FLP admits no approximation solution, the proof can be found by reduction from COVER SET which is not approximated by a polynomial time algorithm with constant factor [109], as opposed to our problem where approximation solutions can be found.

plan that is not very different from S^0 either, or even keep S^0 as a new provisioning plan (no modification is made). However, if the current context is far different from the one at t_0, i.e. S^0 may result in much higher TC than the optimal solution, then we expect a radical change in the service set. Hence the service provider has to decide whether modifications are needed, or how many modifications to be made in function of the difference from the current service set (S_0) and the optimal one.

We first introduce the Hamming distance which qualifies the distance between two sets of replicas. Let S be a service set, we associate S with a vector $x \in \{0,1\}^V$:

$$S \to (x_i), \forall i \in V, x_i = 1 \text{ if } i \in S,\ 0 \text{ if not}$$

Let S' be another service set, denote $|S - S'|$ as the Hamming distance between S and S':

$$||S - S'|| = \sum_{i \in V} |x_i - x'_i| \tag{4.8}$$

that is the number of different replicas of S and S'.

To find a good solution at time $t_0 + 1$, we only consider the service sets that can reduce the total cost proportional to their Hamming distance to service set S^0. Precisely, we only consider P the set of subsets of V such that:

$$P = \{S^0\} \cup \{S \subseteq V, \quad S \neq S^0, \quad \frac{TC^{S^0} - TC^S}{TC^{S^0}||S^0 - S||} \geq \gamma\} \tag{4.9}$$

where TC^{S^0} is the total cost of service set S^0 at the time $t_0 + 1$ and γ is a constant. A new set is allowed only if it could reduce the TC with at least an amount proportional to its (Hamming) distance to S^0. The constant γ quantifies this proportion, and is determined by the service provider's policy. For example, if the policy is "to allow one replica difference for each 5% of cost reduction", then γ value will be 5%. γ is the trade-off threshold between the level of modifications and the reduction of the total cost of the provisioning plan that these modifications bring back.

Let

$$TC_d^S = TC^S + \gamma TC^{S^0}||S^0 - S|| \tag{4.10}$$

We can rewrite P as set of all $S \subseteq V$ such as:

$$\begin{aligned} & TC^{S^0} - TC^S && \geq \gamma TC^{S^0} ||S^0 - S|| \\ \Leftrightarrow \quad & TC^{S^0} && \geq TC^S + \gamma TC^{S^0} ||S^0 - S|| \\ \Leftrightarrow \quad & TC^{S^0} && \geq TC_d^S \end{aligned}$$

We would like to find the service set $S_{min} \in P$ that minimizes the total cost TC^S (Eq. 4.1). However, as P and S_{min} are difficult to evaluate, we focus our attention to S^* which minimizes TC_d^S (Eq. 4.10). The reason is threefold. Firstly, since S^* is the minimum of TC_d, so $S^* \in P$ (TC_d, $TC^{S^*} \leq TC_d^{S^0} = TC^{S^0}$).

Secondly,

$$\begin{aligned} TC_d^{S^*} &\leq TC_d^{S_{min}} && (4.11) \\ \Rightarrow 0 \leq TC^{S^*} - TC^{S_{min}} &\leq \gamma TC^{S^0} (||S_{min} - S^0|| - ||S^* - S^0||) \\ &\leq \gamma TC^{S^0} ||S_{min} - S^*|| \\ \Rightarrow \frac{TC^{S^*} - TC^{S_{min}}}{TC^{S^0}(||S_{min} - S^*||)} &\leq \gamma && (4.12) \end{aligned}$$

So even if S_{min} can further reduce the TC, it needs more modifications and the cost reduction cannot make it to the threshold γ required by the policy (Eq. 4.9).

And lastly, the integer linear program of (4.10) is:

$$\text{Min} \sum_{i \in V} c_i x_i + \sum_{i,p \in V} d_{pi} w_p y_{pi} + \gamma TC^{S^0} \sum_{i \in V} |x_i - x_i^0| \tag{4.13}$$

$$\begin{aligned} s.t. \quad & y_{pi} \leq x_i && \forall i, p \in V \\ & \textstyle\sum_{i \in V} y_{pi} \geq 1 && \forall p \in V \\ & x_i, y_{pi} \in \{0,1\} \end{aligned}$$

where x^0 is the associated vector of S^0.

Since $|x_i - x_i^0| = x_i^0 + (1 - 2x_i^0)x_i$ for any $x_i, x_i^0 \in \{0,1\}$, we can rewrite (4.13) as:

$$\sum_{i \in V} (c_i + \gamma TC^{S^0}(1 - 2x_i^0))x_i + \sum_{i,p \in V} d_{pi} w_p y_{pi} + \gamma TC^{S^0} \sum_{i \in V} x_i^0$$

Thus (4.13) becomes:

$$\begin{aligned} \text{Min} \quad & \textstyle\sum_{i\in V}(c_i+\gamma TC^{S^0}(1-2x_i^0))x_i+\sum_{i,p\in V} d_{pi}w_p y_{pi} \\ s.t. \quad & y_{pi}\leq x_i \qquad \forall i,p\in V \\ & \textstyle\sum_{i\in V} y_{pi}\geq 1 \qquad \forall p\in V \\ & x_i,y_{pi}\in\{0,1\} \end{aligned} \tag{4.14}$$

which is the same form as the integer linear program of (4.2) by changing the parameters. Thus S^* is a static solution of the initial problem presented in section 4.3 with different parameters, which can be efficiently calculated using any FLP approximation algorithm. From these observations, we see that the choice of S^* as a new set of replica is appropriate for both the purpose of reducing the total cost as well as controlling the replica modification. From now on we will refer to (4.10) as *adaptive formulation*, and (4.1) as *classical formulation*.

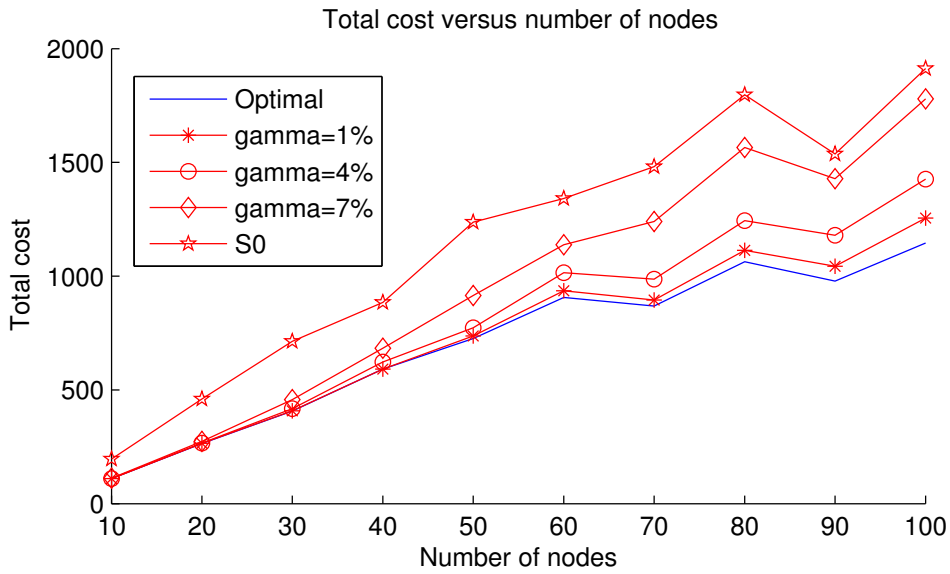
4.6 Experiments

We have conducted numerous experiments to evaluate the impact of our new scheme on the placement of service replicas. For each experiment, we generate a set of random graphs using the Networkx library [5], and run a series of tests on these graphs. These random graphs are generated using different models ($Erdõs-Rényi$, Small-world , $Barabási-Albert$). We observe that the results are similar for all three graph models. In this thesis we present only the results taken from $Erdõs-Rényi$ random graphs as representative. We use Mahdian's greedy algorithm [83] to calculate the service set, which is known to have the approximation factor of 1.861. In this section, we refer to the values calculated from classical formulation as "optimal", and we compare it with the output from our adaptive formulation with different values of γ.

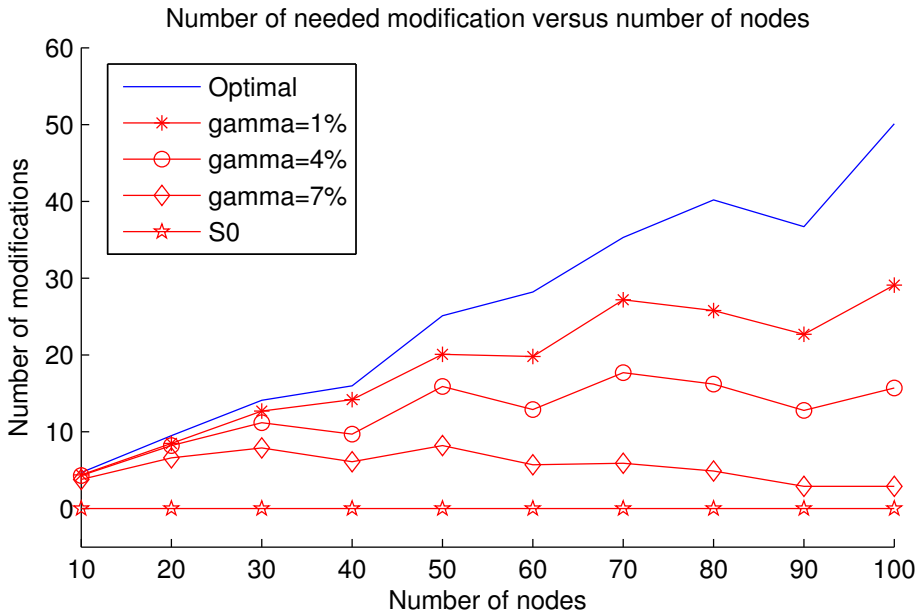
4.6.1 Efficiency evaluation

In this experiment, we compare the result taken from the adaptive formulation and the optimal total cost taken from the classical TC formulation. The results are compared in different schemes (classical formulation, adaptive formulation with different values of γ). In the initiation phase, a random graph with all parameters is generated, and an optimal provisioning plan S^0 is calculated. In the second phase, all the parameters are re-generated randomly, and the provisioning plan S^{opt} is calculated using (4.1). This plan is then compared with the

results obtained by solving the adaptive formulation (4.10) with different values of γ. We use the policy which "allows one modification for each γ (percent) of reduction", where γ is 1%, 4%, and 7% . The number of modifications is determined by the Hamming distance between S^0 and the service set S calculated in each scheme (classical formulation, adaptive formulations).



(a) Total cost



(b) Number of modifications

Figure 4.3: Efficiency evaluation. Each value is taken from 20 random network graphs of the same number of nodes.

As we can see in Fig. 4.3.a, the smaller γ is, the better the outcome is (i.e. the nearer it is to optimal value), and the highest total cost is TC^{S_0}, i.e. without modification, which is actually the case where $\gamma = 100\%$. On the contrary, higher values of γ require less modifications, and the highest number of modifications is the optimal solution which is the case where $\gamma = 0\%$ (Fig. 4.3.b). Indeed, high values of γ strictly restrict the number of modifications to the previous service set, which leads to limited choices, while small γ tolerates more modifications, which leads to values closed to the optimal provisioning plan. We observe that the number of avoided modifications by applying new scheme is more important than the increase of the cost. With $\gamma = 1\%$, 40% of the number of modifications are avoided when compared with the optimal solution while the total cost only increases slightly.

The second remark is that the curves diverge when the number of nodes increases. This is due to the fact that as the size of the network increases, the contribution of a replica to the reduction of service delivery cost decreases, so adding/removing a replica in a large network will have a weak effect on the total cost. As a consequence, it is unlikely that one modification can reduce the total cost to the threshold of γ. Hence, service providers should adjust γ as their need: in a large network, they should allow more modifications (thus, smaller γ) in order to maintain the efficiency of their service replicas. Regardless, while the ratio of total cost with $\gamma > 0$ and the optimal total cost increases slowly (e.g. for $\gamma = 7\%$, the ratio increases from 1.0 at n = 10 to 1.5 at n=100), the ratio between the number of modifications of the two increases very quickly (e.g. for $\gamma = 7\%$, the ratio increases from 1.0 at n=10 to 17.2 at n=100).

4.6.2 Performance stability

In this simulation we evaluate the impact of the adaptive formulation (4.10) on the system in long run. In the previous test we compare the adaptive provisioning plan and the optimal plan with a same plan S^0. In this simulation, we will apply each scheme separately in succession. Precisely we will apply each scheme (optimal, γ = 1%, 4%, 7%) for a hundred of periods on a same graph. In each period, all parameters are re-generated but have the same values for every schemes. In this simulation, the calculated provisioning plan of a period is used as S^0 of the next period. Therefore, S^0 is different in every scheme in every period.

This simulation is to respond to the following concern: if the total cost of current provisioning plan has some distance to the optimal one, could it lead to farther distance of the new plan to the optimal one in the next period? If this is the case, it may lead to an undesirable trend that our results will diverge from the optimal solutions.

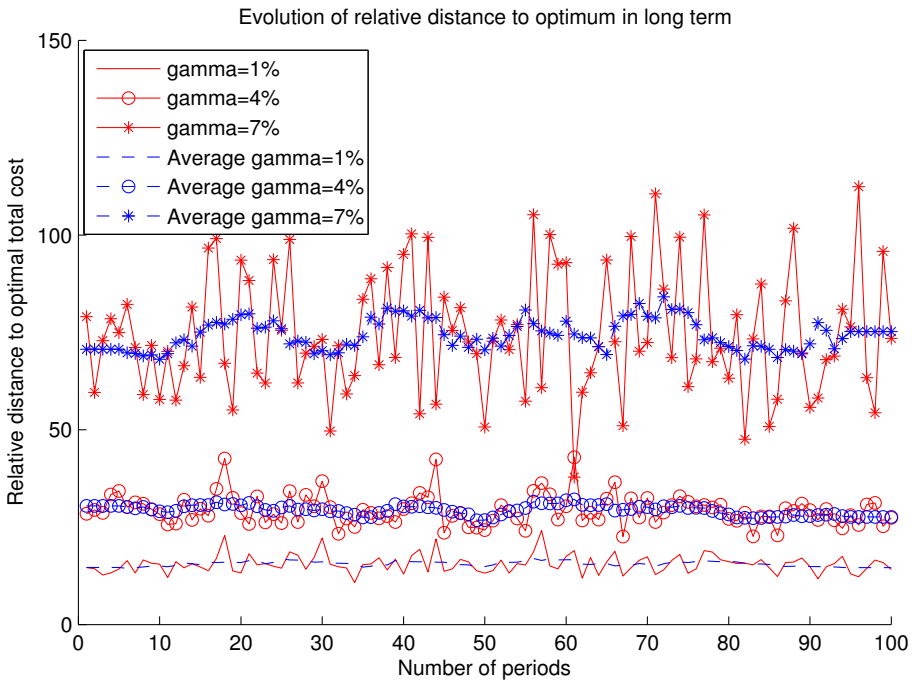


Figure 4.4: Performance stability test: The top three curves in the legend table are values taken at each period, the bottom three ones are average values taken from 10 successive periods (e.g. the value of the curve 'average gamma=7%' at 30th period is the average of 10 values of the curve 'gamma=7%' from 25th to 34th periods)

Starting from a random graph, we run the simulation for 100 successive periods (each period takes the previous service set as input S^0) on random network graphs of 70 nodes. Although the graph topology is the same for all periods, the quantity of request demand, as well as edge length and installation cost change at each period. We repeat this simulation on many graphs and take the average of the total cost. We compare the total cost with the optimal value of each period. The purpose of the tests is to see if after an arbitrary number of periods, the returned total cost diverges from the optimal one or not. Intuitively, since γ limits the number of modifications, the next service set may be "restricted" in the vicinity of the previous service set, thus may get farther from the optimal one. We would like to see if the phenomenon persists in long term. We calculate the relative distance to optimal solution ($(TC^{S^*} - TC^{S^{opt}})/TC^{S^{opt}}$ where S^{opt} is optimal provisioning plan calculated by (4.1) and S^* is calculated by (4.10)). Due to the fact that the total cost varies from one period to the next, we take the average value of 10 successive periods for comparison.

Fig. 4.4 shows that the performance of the algorithm does not depend on the number of periods. We can see that the average value curves are almost horizontal. Although the fluctuation of exact-value curves (top three curves in the legend table) is important, the

average values (bottom three curves in legend table) don't change much. The total value taken after an important number of periods is almost the same as in the first periods, implying that the curve is not divergent. This shows that after an arbitrary number of periods, the result from our scheme remains at the same relative distance from the optimal one. This proves that our scheme is stable over time.

4.6.3 Adaptation to slowly changing environment

In this simulation, we apply our scheme to a particular case where the network environment changes linearly (including edge distance, demand quantity and installation cost). Such changes in environment are possible in short periods of time. We consider that each parameter (demand, cost) of each node has its own time to change tc and speed sc of changing, tc and sc are bounded. In the simulation, every parameter changes with step sc from one period to the next one until the time tc is reached, at that moment it resets tc and sc (drawn randomly uniformly in their bound) and repeat the change process.

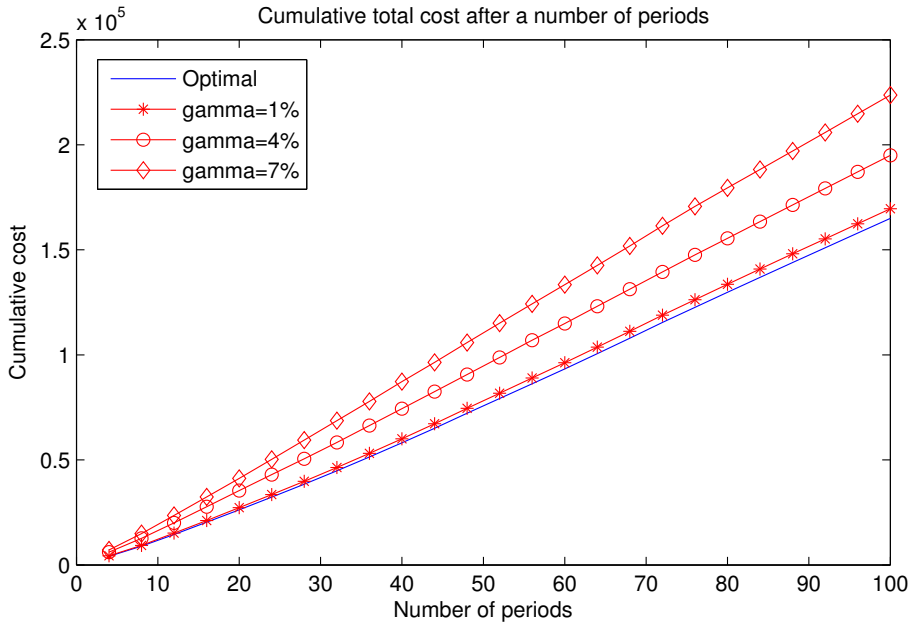
Contrary to the last test where the environment condition at each period is totally random and independent from the previous period, the condition in this simulation strictly depends on the previous one in a linear relation. In this situation, we expect that a low number of modifications in the provisioning plan should be enough to reach a near-optimal provisioning plan. For n-th period, we calculate the cumulative cost, as the sum of the costs that the service provider has to pay up to n-th period (i.e. $\sum_{i=1}^{n} TC(i)$) (Fig. 4.5.a), and the cumulative number of modifications, which is the number of modifications made up to n-th period (i.e. $\sum_{i=1}^{n} Number\_of\_modifications(i)$) (Fig. 4.5.b).

From Fig. 4.5, we observe that the number of needed modifications to maintain the optimal provisioning plan is very high. By contrast, adaptive formulation requires much less modifications while its total cost still remains close to the one of the classic formulation. By sacrificing 2.7% of total cost ($\gamma = 1\%$), we can save up to 35.6% modifications.

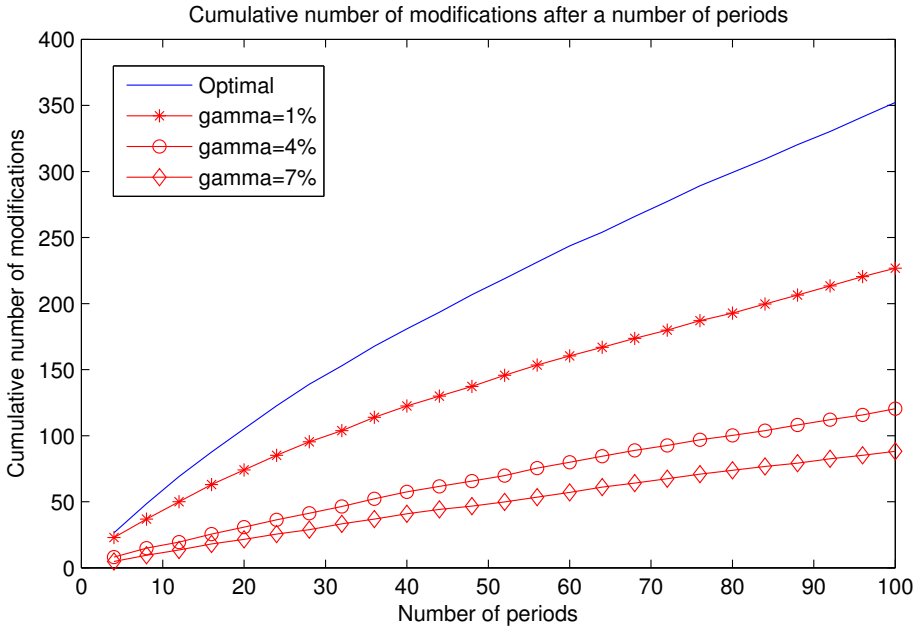
γ	Distance to optimal TC	Reduction of modifications
1%	2.7%	35.6%
4%	18.2%	65.8%
7%	35.6%	75.0%

Table 4.1: Trade-off between the gain in modifications and the loss in total cost compared to optimal solution

One should notice that when $\gamma \to 0\%$, the total cost converges to the optimal value. This means that with very small γ (between 0% and 1%), the number of modifications increases



(a) Cumulative Total cost after a number successive periods in a slowly changing environment



(b) Cumulative number of modifications after a number successive periods in a slowly changing environment

Figure 4.5: Cumulative total cost and number of modifications after a number of periods

rapidly while the cost reduction increases very slowly.

4.7 Conclusion

In this chapter we have presented our study on the problem of placement of service replicas in a cloud to satisfy a foreseen end-users' demand. The problem is for the service provider to propose a solution to the cloud customer that can satisfy the demands of all clients (service's end-users) of the latter with the minimal cost. We have proposed an optimization formulation that restricts the number of modifications on service placement following a high level policy. We argue that to reduce the level of service disruption, the service provider should only allow a number of modifications that is proportional to the reduction of the cost gained from these modifications. The proposed solution introduces a trade-off threshold between the level of modifications to the provisioning plan and the reduction of total cost that these modifications bring back. The threshold should be adjusted depending on the business objective of the cloud customer.

We have conducted several experiments to evaluate this new approach. The results show indeed that the performances of the cloud system implementing our solution conform to the expectation and are persistent in long term. Indeed, the system achieves an optimal trade-off between the cost reduction and the number of modifications. We come to the conclusion that in a slowly changing environment, by lowering the trade-off threshold, service providers can avoid costly automatic modifications in each period, which means less interruption of services, as well as lower cost for such modifications while still maintains a total cost close to optimal. We also note that as the number of replicas increases with the size of the network, the threshold should be dynamically adjusted: cloud customer should tolerate more modifications in large network to find an effective service set.

Chapter 5

Multi-site Cloud Orchestration

5.1 Introduction

Current cloud providers such as Amazon, Rackspace or GoGrid offer their customers the possibility of deploying services among their datacenters around the world. These datacenters are built to allow end-users to be closer to the datacenters and their service delivery points. Customers can use their account to get resources and deploy their services in any datacenter as they see convenient for their end-users. This is an important feature to provide end-users with a better quality of service and therefore increasing their experience.

However, although these datacenters are managed by the same provider, they are autonomous clouds that are independent in terms of resource management, scheduling and provisioning. Each system can be considered as a complete cloud which is capable of providing full cloud capabilities to its customers. Even if customers can deploy their services in multiple datacenters, the deployment is poorly-coordinated with little support from cloud providers. Amazon and Rackspace charge inter-region traffic as normal outbound traffic while GoGrid charges Datacenter-to-Datacenter links per reserved bandwidth. Thus cloud customers have to consider carefully how their service will be deployed to avoid unnecessary costly inter-datacenter traffic. Currently, customers who want to deploy complex services over various locations have to manually calculate a provisioning plan (i.e. choosing basic services, locations and resources quantity) and deploy each basic service separately.

We envision next-generation cloud providers to join together to federate and share their resources as a common cloud infrastructure to provide wider range of service offers in a wider geographical zone. This vision is also shared by many in the scientific community [40, 56, 75, 94, 106, 113, 125]. It is essential for all cloud providers to be able to extend their

offers beyond the boundary of their datacenters. Even Amazon, the foremost and by far the largest and most used IaaS cloud [13], lacks many features that business customers desire, such as the support for SLA and quality of business level. Cooperation with other cloud providers may open services that they couldn't offer to their customers.

As a consequence, the number of locations per cloud will increase covering more ground to better serve users from various locations. However, for customers who want to deploy complex services, such cloud makes manually calculating and deploying provisioning plan tiresome and time-consuming. Moreover, in a constantly changing environment like the cloud, customers need the capability of quickly adapting the provisioning plan of their services to the needs of the clients. Therefore, automatically provisioning services over the cloud spreading different locations will be a powerful tool for service providers to support their customers efficiently.

To fulfil this objective, we have developed the Multi-site Orchestration System (MOSt) to automatically calculate the optimal provisioning plans and deploy the customers' services in the cloud. This system relies on the analytic model and the proposed solution for deploying customer's complex services using resources from various cloud infrastructures presented in Chapter 3.

5.2 System's technical objectives

The main requirements that are taken into account in the design of our system are:

- *Multi-site deployment*: MOSt must be capable of deploying a complex service over different cloud datacenters (called *cloud sites* or simple *sites*) that are geographically distributed. It must provide the customer with a unique access point to submit his service requests and then deploy them.
- *Optimal provisioning plan*: MOSt must provide the customer with the best (optimal) provisioning plan. The customer does not have to worry about the details of the deployment plan if not required. The system must interpret the customer's requests and build the optimal plan with respect to the customer's requirements.
- *Site independence*: MOSt should be independent from any underlying site. This means that the system should not be concerned by the internal details of the sites. Sites can join or leave the system without modifying its architecture, nor that of the system. This stackable design should allow the system to expanse the cloud to third party cloud providers' infrastructure, i.e. deploying complex service using the resources outside the infrastructure of the service provider.

The first two requirements are the main objectives of the system. It is designed such that customer can submit their complex service request at one unique access point and let the system handle all the deployment process from analysing the customer's request to calculating the optimal provisioning plan and finally deploying basic services at the chosen sites. The customer does not need to know or worry about the sites. The service should be deployed following the submitted service description and requirements.

The third requirement is essential to the service provider if resources from other cloud providers are needed. It allows the federation of the infrastructure of other cloud providers without modification to their clouds. Service provider can subscribe to another cloud and deploy a small platform to interact with the local resources without interfering with their functionalities. In our system, the communication with another cloud infrastructure is done via the Site Communication component (See 5.5.2). It requires only a customer account to access to local cloud infrastructure.

5.3 Existing initiatives for IaaS cloud services and resources modelling

There are many efforts to provide interoperability among IaaS clouds. Most of them focus on defining a standard for resources and service description and avoiding vendor lock-in. These standards provide cloud providers and customers with a common language to specify their interfaces. Among the open standards and projects, OVF and OCCI stand as the most widely accepted and adopted ones.

OVF and Reservoir

Open Virtualization Format (OVF) is an open extensible format standard for packaging and distributing virtual appliances[1]. Proposed in 2007 by VMware, Dell, HP, IBM and XenSource (Citrix) in the Distributed Management Task Force (DMTF), OVF is currently at version 1.10 [44]. OVF is designed to provide portability, integrity, and automatic deployment of virtual machines. It allows user to package and deploy their virtual machines on any hypervisor platform without worrying about the compatibility.

OVF can be used to encapsulate one or multiple virtual machines that are inter-dependent. Virtual machines are packaged into an all-in-one, ready-to-run OVF package. An OVF

[1] Virtual appliance is a ready-to-run package that can be deployed directly in a hypervision platform without exterior dependency. It is fined by OVF as "a service delivered as a complete software stack installed on one or more virtual machines. A virtual appliance is typically expected to be delivered in an OVF package" [44]

package is a directory, or a TAR archive [3]. It contains an OVF descriptor file which describes the packaged virtual machine in XML format. OVF descriptor includes metadata fields such as name, hardware requirements (CPU, RAM, Hard disk, network interface, etc.), and references to other files in the package. The OVF package also contains disk image files, manifest, certification or additional resource files, such as ISO images.

The Reservoir project [20, 112, 114] extends OVF format to serve as a communication message template (manifest) between a customer and the cloud platform. Reservoir's objective is to provide customers with a template in which they can specify not only the resources for their services, but also service quality enforcement (Service Level Agreement - SLA) and a set of configuration that should be automatically enforced at the initiation of the services. Although using the same base images, customers can request specific customization according to their needs. These customizations include software parameter setting, network configuration, or even downloading and upgrading software. Therefore, in Reservoir platform, one base image can be used to serve many customers in different contexts.

OCCI and CompatibleOne (CORDS)

Open Cloud Computing Interface (OCCI) [100] is an Open Grid Forum specification founded in 2009. The purpose of OCCI is to provide integration, portability and interoperability between different clouds. Current OCCI provides an API for IaaS clouds [92] which uses RESTful HTTP protocol [91]. OCCI integrates business aspect into the cloud manifest; a new extension was proposed to support Monitoring, SLA and Negotiation.

In OCCI RESTful API, resources are identified by an URL; user can interact with the resources using a HTTP requests (POST/GET/PUT/DELETE) (Fig. 5.1. The three core resource types defined in OCCI are Compute, Storage and Network; OCCI Infrastructure extension adds NetworkInterface and StorageLink sub-types (Fig. 5.2). Resources are linked together following user's application design.

CompatibleOne project [40] proposes CompatibleOne Resource Description System (CORDS) an extension of OCCI model. Customers express their provisioning requirements in a XML file (CORDS manifest). All elements of CORDS are represented using an OCCI category. All categories are directly related to the standard OCCI core type and consequently inherit the attributes from this base type. The Manifest categories defined in CORDS and their relations are illustrated in Fig. 5.3. CORDS Manifest is the outermost category, used to define a particular cloud service which consists of nodes, configuration actions, accounting and security aspects. Nodes regroup all resources, both physical and virtual. Node can be described in terms of infrastructure and software requirements, or in the form of a manifest.

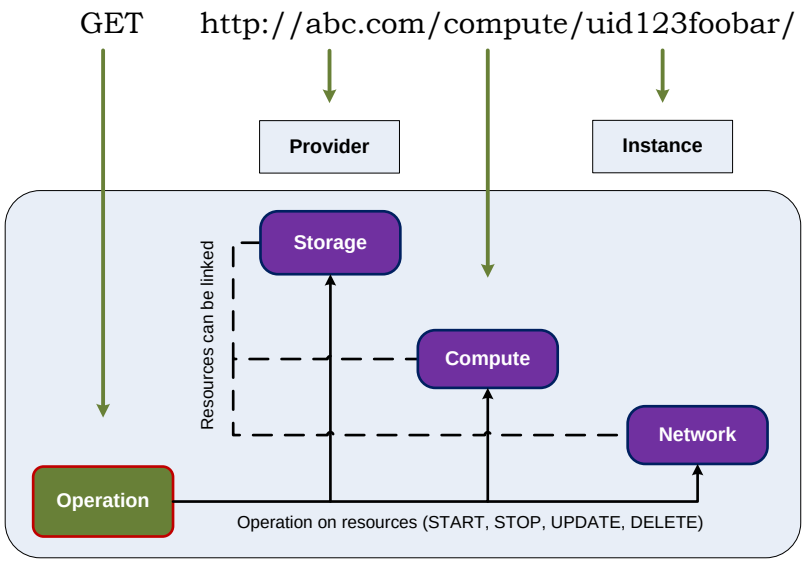


Figure 5.1: OCCI RESTful API. Source: [90]

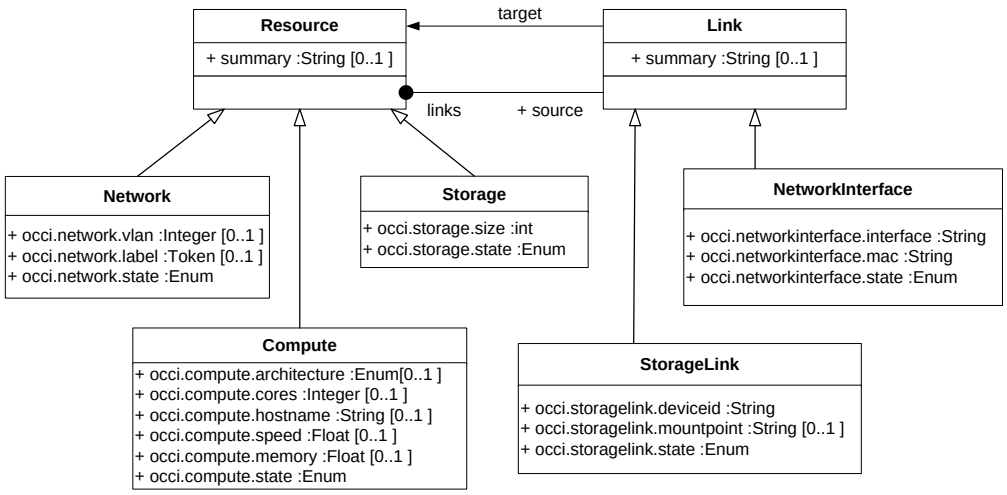


Figure 5.2: OCCI Infrastructure types. Source: [92]

Other categories in CORDS are added to provide service management (Plan, Service, Contract, Instruction, etc.), provisioning system management (Operator, Publication, Provider, Authorization, etc.), pricing (Price, Transaction, Invoice), SLA support (Agreement, Term, Guarantee, Penalty, etc.) or Monitoring (Session, Monitor, Packet, Metric, Alert, Event, etc.). CORDS also uses RESTful API proposed by OCCI.

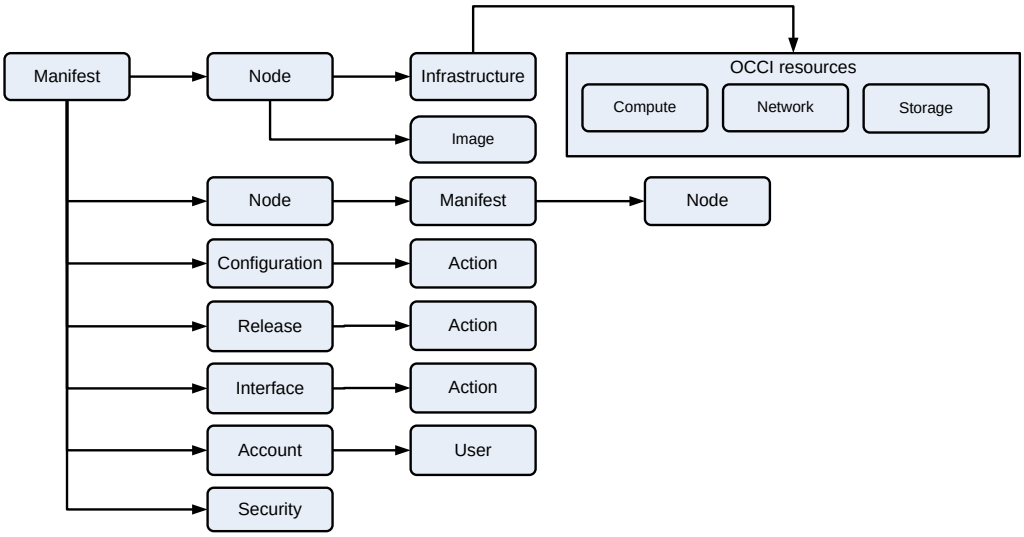


Figure 5.3: CORDS Manifest categories. Source: [41]

Other proposals

Topology and Orchestration Specification for Cloud Applications (TOSCA [101]) is proposed by OASIS consortium to enable the interoperable description of application and infrastructure cloud services. TOSCA focuses on the relationships between components of a complex service, and the operational behaviour of these components (e.g., deploy, patch, shutdown). Complex services are described in form of workflow in which nodes represent components/appliances and links represent relation between nodes. TOSCA uses WSDL [34] and XML for its declaration.

Nimbus project [86, 95] aims to provide a context-aware computing platform that can initiate a cluster of virtual machines at once. Users specify the cluster of virtual machines and the context of each virtual machine. The Nimbus manifest allows customizing the role of every virtual machine and the configuration of its application according to the role of the virtual machine. At the initiation of each virtual machine, a pre-installed script inside it will contact the context broker to configure the virtual machine specified in its context.

AWS CloudFormation [1] was launched by Amazon in 2012 to create, manage, provision and update a collection of resources in AWS cloud. AWS CloudFormation is an Amazon's offer to automate the deployment and management of users' virtual appliances on AWS cloud, which is up till now almost manually done by users. Although not an open standard, the dominant position of Amazon's AWS cloud makes it too important to be ignored. AWS CloudFormation helps user create his own JSON [43] template to describe resources and associated dependencies or runtime parameters for the applications. AWS CloudFormation

will solve the dependencies and initiate and configure the resources. AWS CloudFormation also notifies users when each part of the collection as well as the whole collection are up and ready to run. The whole collection of virtual machines is considered as one unit; therefore users can start or stop an entire environment in a single operation.

5.4 Complex service model

In this work, we reuse the service model presented in Chapter 3. In this model the cloud is represented by a substrate graph. Each node of the graph is a cloud site providing a set of basic services with certain usage price. Initiating a service on a node implies a cost that depends on the service and the resources used by this service (e.g. number of virtual machines for a normal computing service or size of the storage for a storage service). The edges represent the network resources between these nodes. Like the nodes, every edge has its own usage price (e.g. per traffic volume or per reserved bandwidth prices).

A complex service is specified as an abstract graph (called *service graph* or *request graph*). Each abstract node represents a basic service and the quantity of resources needed for this service. The abstract edges represent the network resources used to communicate between abstract nodes (traffic volume or level of reserved bandwidth).

Deploying the complex service in the cloud means that the service provider has to find the substrate nodes that can host the abstract nodes, i.e. provide the same services with the same requirements as defined in the abstract graph. If two substrate nodes are chosen to host two abstract nodes, then the service provider has to allocate resources of the path between these substrate nodes to supply the communications between the two abstract nodes as indicated by its abstract edge.

The deployment of a complex service generates a cost that is associated with node resources on chosen substrate nodes and network resources on chosen substrate paths. As usual, the customer expects the service provider to provide him with a provisioning plan with as low cost as possible.

Considering the following scenario presented in Chapter 3 in which a media company wants to deploy an Online Video Streaming service destined for the French market. The defined request specifies a Storage node to stock the media content, a Transcode node to adapt the media to his clients' terminal supports, and a Portal node as detailed in Table 5.1.a. Since the videos are converted at the Transcode node and streamed directly to clients' terminals in France, it is required that the Transcode node is installed in France. This makes it easier to ensure the quality of the services delivered to the clients in France. The Storage

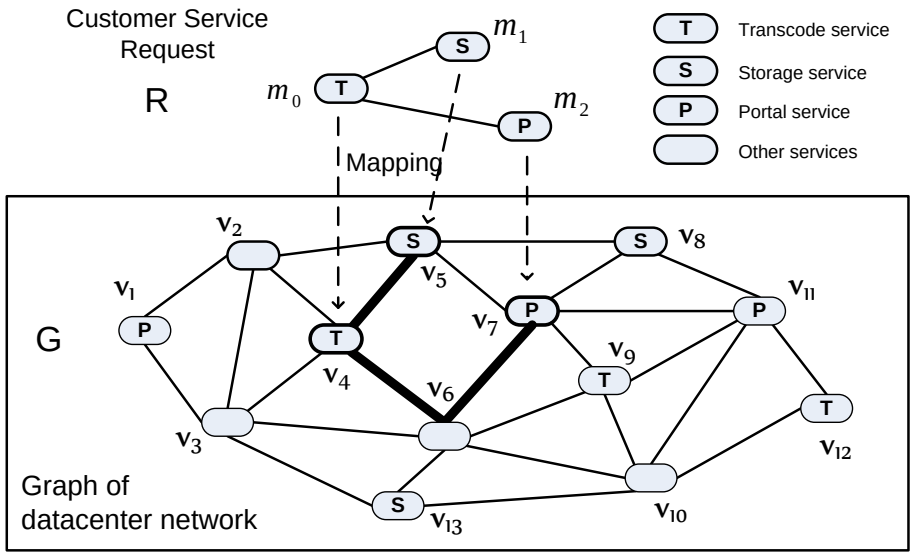


Figure 5.4: Mapping Online Video Streaming service graph R into substrate network graph G

(a) Complex service requirement

Request components	Quantity	Requirements / Attributes
Storage Node	500 GB	Availability :98%
Transcode Node	10 instances	Location : “eu:france" Availability : 95%
Portal Node	2 instances	None
Link Storage - Transcode	40 GB/hour	None
Link Portal - Transcode	5 GB/hour	None

(b) Substrate network

Storage hosts	$0.001 / GB per hour
Transcode hosts	$0.30 per instance per hour
Portal hosts	$0.50 per instance per hour
Links	$0.10 / GB per hour

Table 5.1: Online Video Streaming Request of Fig. 5.4

node does not have any constraint on location, since clients do not have direct connection to it, but only via the Transcode node. However, it requires very high availability and an important traffic link to the Transcode node. Finally, the Portal node serves as clients' access and control point. It does not requires either location restriction nor high availability but only a low traffic link to the Transcode node. Two instances of this node are required for better redundancy.

Suppose that the cloud provides these services with the pricing as detailed in Table 5.1.b.

If we deploy the complex service as Fig. 5.4, then the total cost (per hour) of the deployment is (remark: the path $V_4V_6V_7$ is used for the communication m_0m_2):

$$0.001 \times 10000 + 0.30 \times 10 + 0.50 \times 2 + 0.10 \times 50 + (0.10 + 0.10) \times 5 = 20.00\$$$

As stated in Chapter 3, finding the optimal deployment is NP-hard. Hence we proposed the Incremental Graph Mapping (IGM) algorithm to find an approximate solution for the deployment. This algorithm has been implemented in the system as presented in the following sections.

5.5 Proposed Multi-site Orchestration System architecture

5.5.1 System overview

Our system design follows the requirements identified in the previous section. The substrate nodes in the network graph are *sites* located in different geographical locations. Each site is a full IaaS cloud operated by a *Site Manager* (OpenStack [6] in our implementation) which has its own service catalogue and price. Node resources are Virtual Machines and Storage Volume that cloud customers may request, and their price depends on the node, service and the type of the resources (Virtual Machine/Volume). In our design, the sites are independent in terms of resource management. Each is capable of deploying services via its own API.

The system is responsible for the global provisioning only, meaning it will decide which component will be initiated at which node. At the core of our system, *Global Orchestrator* takes customers' requests, builds an optimal provisioning plan and sends commands initiating each request component (abstract node in the analytic model) to the chosen site using its API. How Site Manager handles the request component (local provisioning) depends on its own implementation. Therefore, sites may use different scheduling mechanisms to deploy service component without the awareness of the MOSt system and other sites (as explained in 5.2). This preserves the independence of the sites. If a site is built upon another cloud provider's infrastructure, Site Manager's role is reduced to simple a wrapping interface to the local cloud.

5.5.2 System components

Fig. 5.5 illustrates the global design of our architecture. There are 5 main components in our architecture:

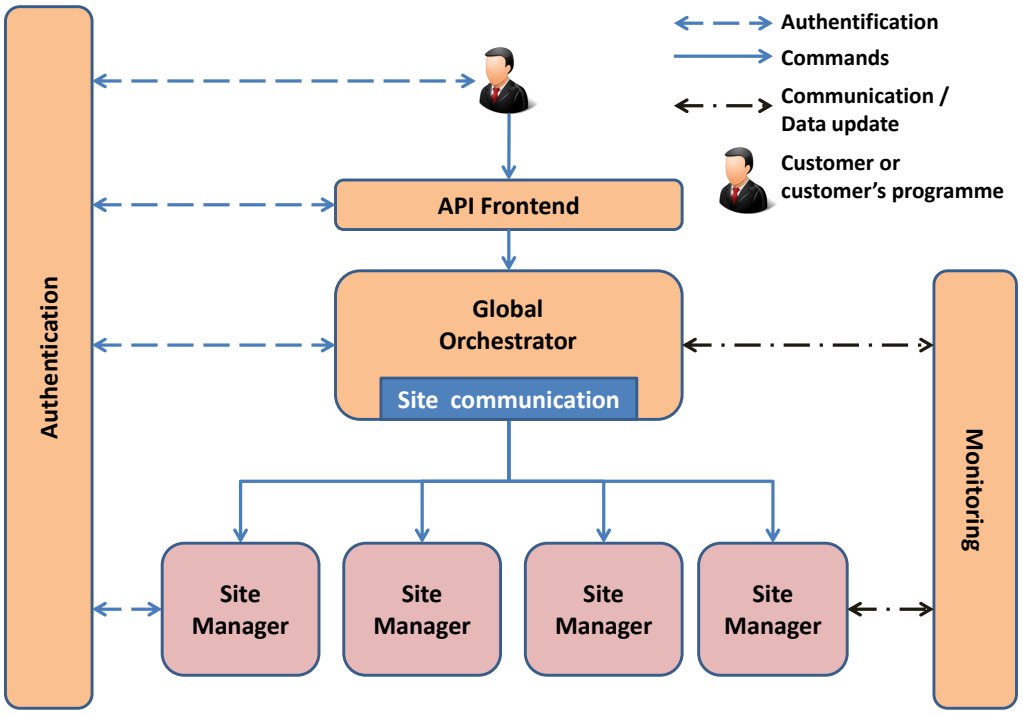
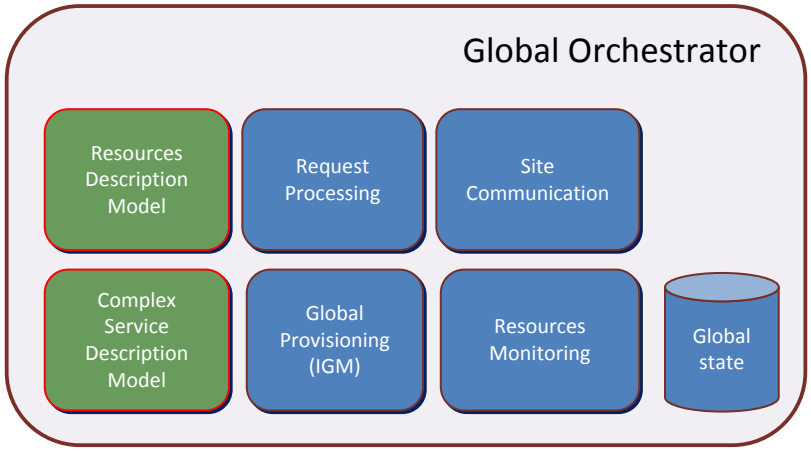
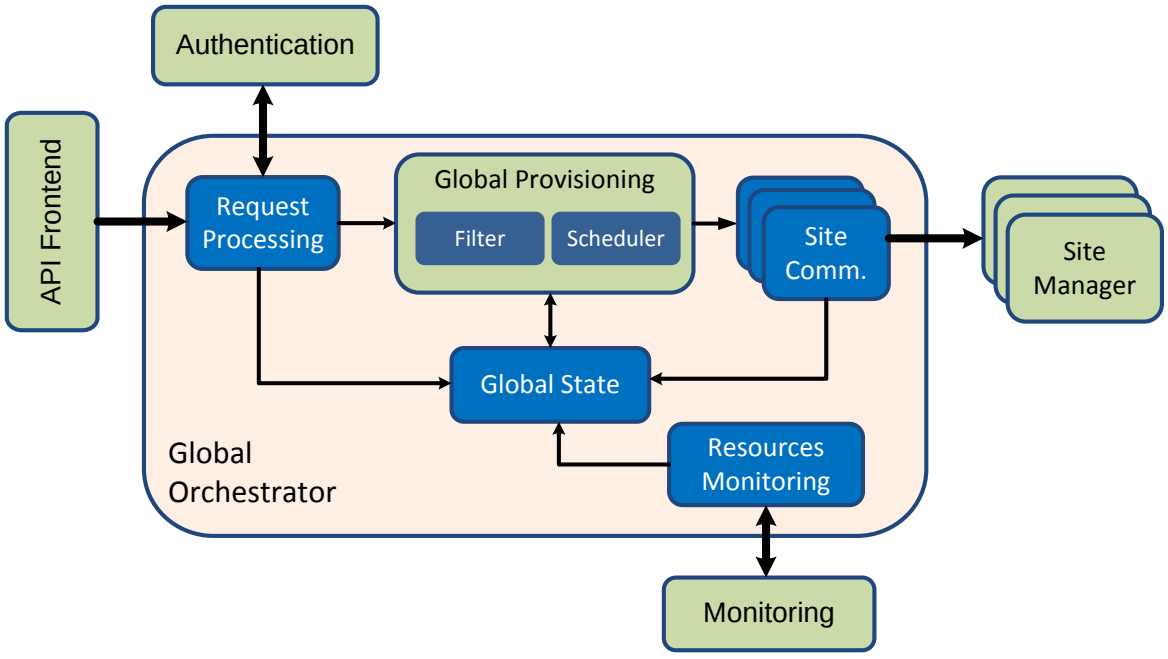


Figure 5.5: Global architecture

- *Authentication*: provides identification functionality and manages tokens and admission control. Authentication creates one token per user's session. This token authenticates all transactions during the user's session and is used to identify user at other components. Only one token is validated for one user at all time. If the user requests another token, the latter will overwrite any existing one. If no token request is made, a token will automatically expire within 24h. The Authentication certificates user via the active token, which indicates the right of the user and the resources to which he could access.
- *API Frontend*: receives user's (or user's programme's) commands. API Frontend also pre-processes user's command by verifying its token, his right, and the command's syntax.
- *Global Orchestrator*: the core of our system, responsible for building provisioning plan and commanding the underlying sites. Global Orchestrator builds a request graph based on user's request. It uses IGM algorithm (See Section 3.8) to calculate the optimal provisioning plan. Finally the Global Orchestrator sends the requests to the correspondent sites to initiate the service components.
- *Monitoring*: provides and aggregates information on the current state of each site and



(a) Global Orchestrator components



(b) Logical blocks

Figure 5.6: Architecture design

resources including customer's resources usage and performance.

- *Site Manager*: each site is operated by a Site Manager. Site Manager is autonomous, capable of providing a complete cloud functionality such as resource management, networking, security, and monitoring. In our system we use OpenStack [6] as Site Manager. If a site is built on third party provider's cloud infrastructure, then its role reduces to simply providing a translation interface with a (local) customer account to get access to its local cloud resources.

The Global Orchestrator (Fig. 5.6) its itself composed of:

- *Resource description model*: a model for representing resources in the cloud. The basic node resources in the system is the Virtual Machine (VM) and the Storage Volume, each is configured by number of Virtual CPUs (VCPU), RAM, Hard disk and images which provides application and integrated operating system for VM and storage size for Volume. Link resources are determined by the traffic volume. While OpenStack supplies the available capacity of every physical server, such information is too large for inter-site communication. In our system, Computing resources are defined by the types of VM to set up, each type is a pre-defined configuration of VMs (VCPU, RAM, Hard disk). This approach helps aggregate information for resource management (see Resource Monitoring below). Each type of resources is associated with a semantic description that features the propriety of the correspondent resources (i.e. server availability, data durability, location, etc.).
- *Complex service description model*: a model for representing customers' demands. Following the model presented in Section 5.4, customer's resource request is in a form of a complex service composed of nodes and links, each is specified in terms of service types, required resources respecting the Resources description model. Extra information such as access key and security definition are also included.
- *Request processing*: The entry point of the Global Orchestrator, Request processing analyses requests received from API frontend. It also communicates with the Authentication module to verify if the user has the right to access the resources. It also builds a request graph corresponding to the request.
- *Global provisioning*: Global provisioning implements IGM to calculate the optimal provisioning plan for the request based on the request graph made by Request processing subcomponent. Two operations are successively applied: *filter* which selects the admissible basic services and resources for the user's request and *scheduler* which calculates the optimal placement for the service components. Global provisioning then commands initiating the service components to the sites chosen in the provisioning plan through the Site communication module.

- *Site communication*: Site communication handles the communication with every underlying site. It issues commands to every site regarding the provisioning plan built by Global provisioning. In case of heavy workload, several instances of Site communication will be generated. In practice, Site communication is separated from other subcomponents and installed in several virtual machines using a common message queue which stores the messages from Global provisioning. The number of Site communication instances (number of virtual machines) increases following the number of messages in the queue.
- *Global State*: Global State is a database that maintains information on users' resources metadata and current state of the entire cloud, including characteristics, properties, service category and prices for each type of resources of every site.
- *Resource monitoring*: observes the available resources from sites. Resource monitoring uses information from Monitoring component to regularly update Global State (every 30 seconds in our implementation). Only maximal supported number of each VM/Volume types are provided.

5.5.3 Message flow

Our system uses RESTful HTTP protocol [52] to communicate between users and the system as well as among the components. Fig. 5.7 illustrates the process for creating/modifying a complex service. The message flow is as follow:

1. At the beginning of each session, user identifies himself at the Authentication. The Authentication component issues a random token to identify the session. The token is used by the user to authenticate his commands. New session's token will replace the current token of the user if he has any, meaning the closing of any session that user's created. All messages exchanged among components include the new token to authenticate themselves.
2. User sends a command to API Frontend using his new token.
3. API Frontend makes a quick check of the user's message and contacts Authentication component to verify the validity of the token and the user's right.
4. If verification passes, API Frontend relays the message to Global Orchestrator.
5. Request processing subcomponent checks with Authentication to verify which resource and service types are admissible to the user and creates a request graph for the user. IGM is then used to calculate the optimal provisioning plan with the respect to user's request.
6. After the provisioning plan is decided, Global Orchestrator sends commands to every

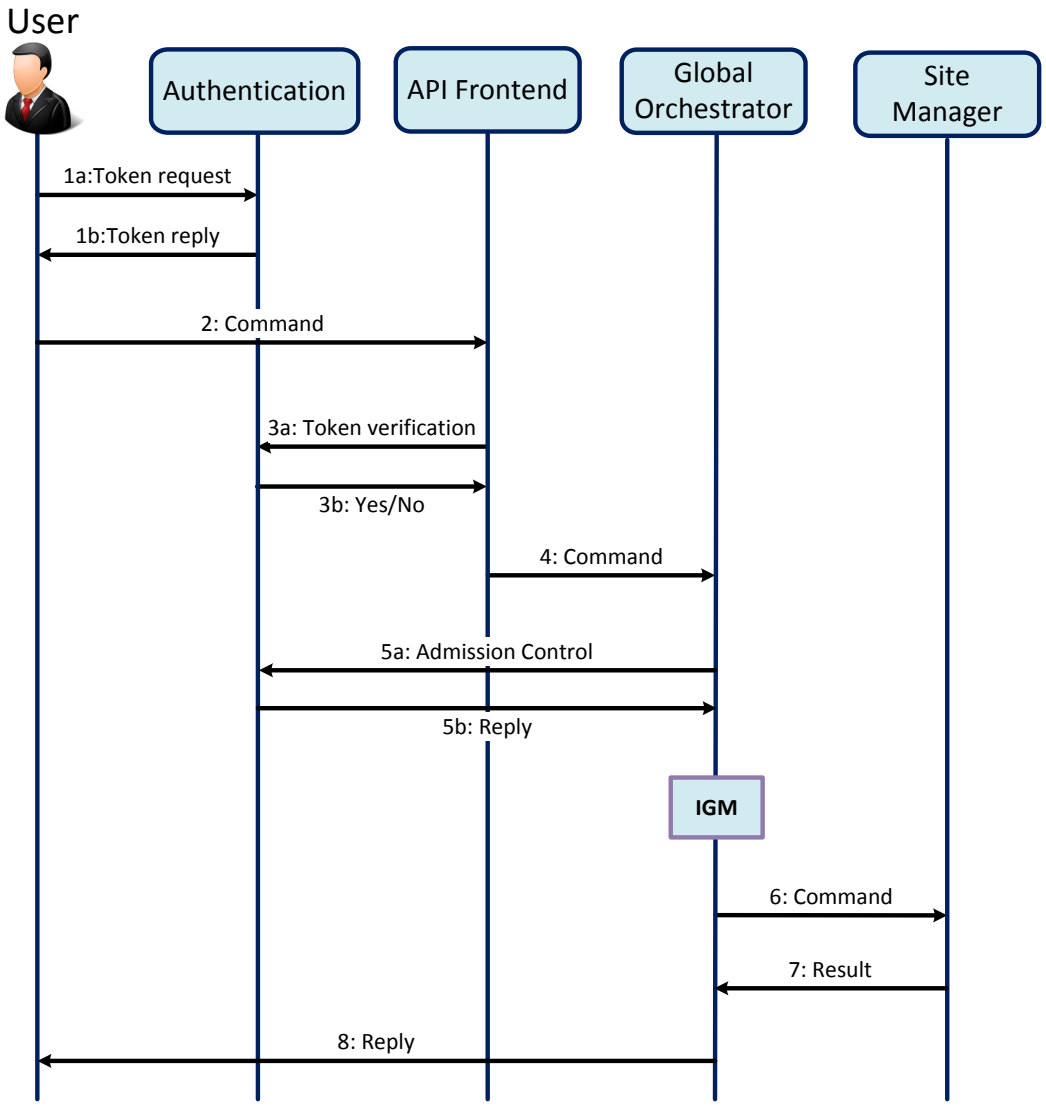


Figure 5.7: Command sequence

concerned Site Manager to deploy the services.

7. If no error arises, the Site Managers report the success to Site Manager, which updates Global State database.
8. Finally, Global Orchestrator sends response to user with the details of deployed complex service.

Due to the high delay to initiate the virtual machines which may take several minutes depending on the required number of virtual machines, Global Orchestrator will not wait for the final result of the initiation process before responding to user. Instead, it will respond to user with an URL pointing to a log page on which the process is detailed.

5.6 Testbed implementation and deployment

We have set up a cloud testbed for the experimentation which is composed of 2 sites (“lab:office" and “lab:rack"), each site is composed of 2 servers Dell T620 (2 x Intel Xeon 4 Cores 2.13GHz, 16GB RAM, 1Tb Hard disk) operated by OpenStack (Essex version) as Site Manager (Fig. 5.8).

Two servers in each site are connected directly by a Ethernet cable in a private 10.0.0.0/16 subnet. We use the Virtual LAN (VLAN) in this subnet to isolate the users' spaces, each user is associated to a VLAN. Remark that 10.0.0.0/16 subnet is not the private network that we associate the virtual machines with. It is merely a subnet on which we create Virtual LAN. The private network that is associated to all the virtual machines is 10.1.0.0/16 (See next section for more details). One of two servers in each site is connected to the IBISC Laboratory network and plays as the gateway to the Internet and the API portal of the site.

We install OpenStack on all two servers of each site, making them a full cloud. We installed the MOSt system in the gateway server of the “lab:office" site, which listens to the its dedicated port 8080. For Authentication component, we use Keystone (See next section) which registers the users of the cloud. Monitoring component is a simple script that reads the OpenStack database in each site and reports the global available resources to MOSt every 30 seconds.

5.6.1 OpenStack overview

OpenStack is a cloud toolkit designed to build a private IaaS cloud using KVM hypervisor. The main components of OpenStack are Keystone - an identification service, Glance - an image services, Swift and Cinder (Nova-volume) - storage services (providing Stor-

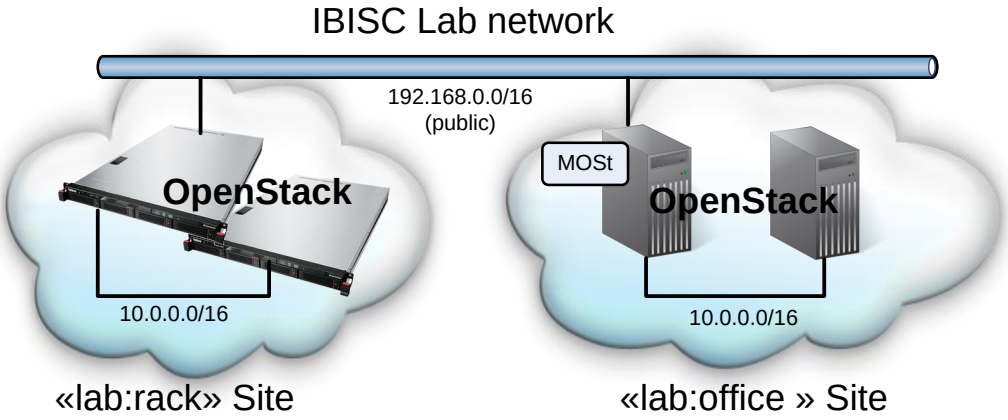


Figure 5.8: Testbed platform

age Volume) and Nova - a computing service (providing Virtual Machines). Nova also provides core networking functionality (Nova-network), scheduling and API , all communicating through a message queue and a database to store system state (Fig. 5.9). By default, resources provisioning follows one of these algorithms: least-loaded (least-loaded physical server is chosen), random (physical server is chosen randomly), in-zone-random (physical server is chosen randomly within in a specific group) or weight scoring (physical server is chosen based on a weight scoring system).

An OpenStack tenant is defined as *project*, determining proprietary space for each user within cloud. Keystone manages the user and tenant registries. It associates each user with a token to authenticate his session. The token is issued at user's request and expires within 24h.

OpenStack Nova-network creates a private VLAN network and attach it to a physical link between the servers (in our platform, the physical link subnet is 10.0.0.0/16 while the private VLAN network is 10.1.0.0/16). The every VM's network interface is given the addresses from this network by the Nova-network. Each tenant (user) is associated with a VLAN. Within his VLAN, a user can create, release VMs and Volumes as he wants, totally separated from other users, with his own firewall and security set-up. User can also allocate a public IP address and attach it to a VM. Using public IP addresses is the only way to connect to the VMs from outside. OpenStack Nova-network plays the role of a Network Access Translation (NAT) which translates a public IP address to a private IP address.

Adding more physical resources is easy with OpenStack; it is simply done by installing Nova or Swift in the new server. However, there are two centralized points in OpenStack:

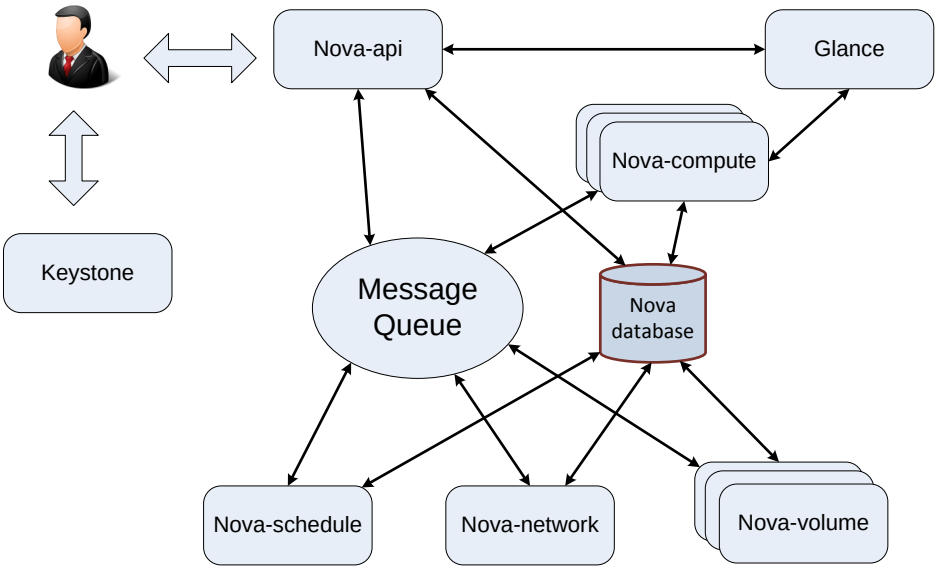


Figure 5.9: OpenStack overview

the message queue and the network manager which are the critical points of OpenStack.

Still, OpenStack is a very good choice to operate a cloud site. Its user isolation fits nicely with our design. It allows VMs of the same users communicating in the same manner as if they are in a LAN network and are separated from others. It makes perfect choice to host an abstract node in the customer's request as presented in Section 5.4.

5.6.2 System implementation

Scenario and experimentations

In our scenario, the cloud is composed of 2 sites, each is a datacenter operated by OpenStack which provides full cloud capabilities. Each offers it own services. The cloud resources are virtual machines (VMs) that provide computing resources through the use of images. Customers can precise two attributes at their specification: location and availability (uptime rate). Any host that satisfies the type and its attributes of a required service can host this service.

Cloud platform

Our pricing model is influenced by Amazon AWS[2] and Rackspace[3] which charge computing resources by hour and inter-regional traffic volume. We define 3 services in the two sites. Two services are offered by “lac:office" site, one with availability 95% and one with availability 90%. The other site, “lab:rack", offers only one service, which availability is 90%. All sites charge the same amount of $0.10 per GB of outbound traffic per hour (no inbound traffic is charged), which can be considered as $0.10 per GB per hour on every link among sites.

Service	Site	Characteristic	Cost
Service 1	Site 1	“Location":“lab:office" “Availability":90	Tiny instance: $0.4 Small instance: $0.5 Medium instance: $0.7 Large instance: $1.0
Service 2	Site 1	“Location":“lab:office" “Availability":95	Tiny instance: $0.8 Small instance: $1.0 Medium instance: $1.4 Large instance: $1.8
Service 3	Site 2	“Location":“lab:rack" “Availability":90	Tiny instance: $0.3 Small instance: $0.4 Medium instance: $0.6 Large instance: $0.8
Link between any two sites: $ 0.10 per GB of traffic per hour			

Table 5.2: Cloud set-up

Our cloud offers virtual machines with four different sizes: tiny (1 x Core 1 GHz, 500 MB RAM, 1 GB Hard disk), small (1 x Core 2GHz, 500 MB RAM, 2 GB Hard disk) medium (2 x Core 2GHz, 1 GB RAM, 5 GB Hard disk) and large size (2 x Core 2GHz, 2 GB RAM, 20 GB Hard disk).

Customer’s request

The customer request is a complex service which is composed on two nodes: “alpha" and “beta" as detailed in Table 5.3. At first we make a request for a small link alpha-beta (0.5 GB per hour). Later, we increase the link to 10 GB per hour.

[2]See http://aws.amazon.com/ec2/pricing/ and http://aws.amazon.com/s3/pricing/
[3]See http://www.rackspace.com/cloud/public/servers/pricing/

The mapping cost of a deployment solution is the cost of using the entire resources in the cloud, including resources at nodes and links. It is the sum (per hour) that the customer has to pay to the Service Provider should the requested complex service be deployed.

5.6.3 Communication Protocol and API

We extend the resources description used by OpenStack to include the complex service and service description. For communication between user and API Frontend as well as among system components, we use REST HTTP protocol [52] with content in JSON format [43]. The specification of user's command API of our system is detailed in Annexe 1.

5.6.4 Complex service deployment example

We launch the command requesting the complex service detailed in Table 5.3. Annexe 2.A details the request message. The syntax of the command is:

```
POST /<tenant_id>/compound_app
```

Request components	Quantity	Requirements
Node “alpha"	2 tiny instances	Location : “lab:office" Availability : 95%
Node “beta"	1 tiny instance	Availability : 90%
Link “alpha" - “beta"	0.5 GB per hour (Later: 10 GB per hour)	None

Table 5.3: Request example

The *tenant_id* is the unique number for the tenant (customer's complex service ID). The header of the message contains the token (X-Auth-Token) which is given to the customer to identify the current session. The content is specified in JSON format [43]. The request is read straight-forward:

It is a request for a complex service (*compound_app*), which consists of 3 nodes: one with name “transcode", one with name “mserver" and one with name “portal". The nodes request service of type computing, using the tag *server* to detail the size of the VMs (*flavor_name*), their quantity (*quantity*) and their images (*image_ref*). The *description* tag is used to list all requirements of each node and link (“location" and “availability").

When Global Orchestrator finds a mapping solution, it returns with a CREATED code as illustrated in Annexe 2.B. The *Location* tag contains the URI pointing to a log page that

updates the status of all components of the request. The URL is actually in format:

```
http://<API Frontend address:port>/<tenant_id>/compound_app
```

It is also the URL used in GET compound_app command that the user uses to get details of his complex service. When all components are successfully deployed, the log page changes the complex service's status from PENDING to ACTIVE and shows details of every component in the request. Table 5.4.a summarizes the result of the request which link alpha-beta is fixed to 0.5 GB per hour. Since the traffic between the two is light, it is more economic that node beta is placed in Site 2. When we increase the traffic between the two nodes alpha and beta to 10 GB per hours, the system initiates the virtual machine of beta in the same Site as alpha (Table 5.4.b), since the traffic inside the sites is free of charge.

(a) Link alpha-beta = 0.5 GB

Node name	Site	VM	IP address
“alpha"	Site 1	“alp_0001"	10.1.42.18
		“alp_0002"	10.1.42.19
“beta"	Site 2	“bet_0001"	10.1.0.99

(b) Link alpha-beta = 10 GB

Node name	Site	VM	IP address
“alpha"	Site 1	“alp_0001"	10.1.42.18
		“alp_0002"	10.1.42.19
“beta"	Site 1	“bet_0001"	10.1.42.20

Table 5.4: Complex service deployment result

Remark that the IP addresses shown in Table 5.4 are private addresses. To connect to these VMs, user has to request at least one public IP per node and associate it with one of the VMs.

5.7 Discussions

In our implementation, we use a simple semantic service and resources description model. To better collaborate with other cloud systems, we envisage to use open standard such as OVF or OCCI. These standards, however, are currently not able to describe the complex service graph as we desire. Therefore, there is a need to extend these models.

The current implementation of our system does not provide business support, automatic service configuration nor automatic elasticity. These aspects are currently treated in the

ITEA2's Easi-clouds project. At current state, several proposals are investigated in the project to handle these features. We will address the business support of the system in the future work.

5.8 Conclusion

In this chapter we proposed the Multi-site Orchestration System to deploy complex services in a cloud that covers several datacenters. The system uses IGM algorithm to calculate the optimal provisioning plan and communicates with the Site Managers to deploy the complex service. Additional components are added into MOSt to configure the virtual machines and establish node connections once they are initiated.

Using the system, we successfully deploy complex services on different datacenters of the cloud. MOSt does not directly involve into the datacenters and does not require special requirements for datacenters to work. Therefore, it preserves the independence of datacenters, which allows it to deploy complex services in other clouds using ordinary customer account.

This solution is part of the Easi-clouds project in which it is used as an Executioner to deploy and adjust complex service following decision made by service quality control.

Chapter 6

General Conclusion

Cloud Computing allows customers to set up and operate their services easily and cost effectively. In Cloud Computing, cloud providers are responsible for service deployment and infrastructure management; cloud customers can specify their services and get them deployed into the cloud within brief delay. Customers can scale resources up or down following their needs and pay only for what is actually used. They can deploy services that are already provided by their cloud service provider or create their own complex services mashing-up service components already offered by their provider.

To fulfil these capabilities, there is a need to rehearse existing cloud infrastructures to allow the automatic deployment of the components of the complex services in various datacenters. Hence, there is a need to take benefit from different underlying cloud providers' infrastructures and pricing schemas to find an even better deployment plan. Each service component has its own requirements; therefore it is not efficient that service components that have low quality requirement share the same infrastructure as the components with high requirements (which are more costly). Moreover, one cloud provider may not be able to satisfy the requirements from all his customers, because of the large types of demands in term of quality, cost or localisation (outside his datacenters' location e.g. a country where the provider is not installed). In our vision, future Cloud Computing sees cloud providers cooperate and share their resources in a federated environment to better serve their customers. Using resources outside his datacenters' boundary bring many advantages to a cloud provider: the latter can outsource part of the requests, redirecting them to other cloud providers. However, in current state, there is little interoperation or collaboration between different clouds, or even between datacenters of a same cloud. Customers have to manually calculate a provisioning plan and request the service provider to deploy it. Such laborious work cannot cope with the change in the business. Therefore, automatically calculating a

provisioning plan and deploying it in a short time is a real need from customers to take benefit of the multi clouds environment. It is the objective of our work in this thesis.

Our first step to reach this objective was to model what is a complex service and how to capture the customers' requirements for this service. Our model is graph-based in which service provider's infrastructure is modelled as a substrate graph. By representing the complex service as a graph of its components, the problem of provisioning plan is to find the best mapping of the complex service graph into the substrate graph of the service provider. Unfortunately, we found out that the optimal provisioning plan calculation (in the general case) is computationally impossible. We therefore proposed the Incremental Graph Mapping (IGM) algorithm to find an approximate solution for complex service deployment. Our algorithm can find an optimal solution for complex services in forms of path, star (hub & spokes) or tree topology and an approximation solution for complex service in form of general graph. We also evaluate the performance of our solution in simulation. Compared to ViNEYard, IGM provides better solution and consumes much less resources (time and computing).

As the complex service is deployed in order to serve numerous end-users, the deployment plan should take into account the demand load of the end-users. With this in mind, deploying only one instance of the complex service may not be sufficient to fulfil all end-users accessing the cloud infrastructure from different locations. In this thesis we proposed a solution inspired from "Facility Location Problem (FLP)" resolution and aims to deploy a set of service replicas in the infrastructure to satisfy the end-users from different locations. FLP is designed to solve the problem of placing static facilities like store, distribution centers. Facility Location is only concerned with static solution where little or no modification is expected in long term. Hence it is unable to respond to the constantly changing context of Cloud. To make use of FLP's results, we proposed a schema to reinforce the FLP model to address the specificities of the cloud computing. Our schema takes into account the need of modifying the current provisioning plan to reduce the costs of deployment and re-deployment. Following a simple rule, the proposed solution can greatly reduce the deployment cost.

Finally, we have proposed the Multi-site Orchestration System (MOSt) to support the automatic deployment of complex services over different underlying cloud computing infrastructures. In our vision of a federated environment, automatically deploying service components on different cloud infrastructures (sites) is primordial. As cloud service providers federate their resources, the cloud becomes a heterogeneous environment with services of different quality and prices at different sites. Cloud service providers would need a system that coordinates the complex service deployment over these sites. With this purpose

in mind, we proposed a system that can calculate the optimal provisioning plan and initiate resources for every service component in the identified target site. The system preserves the independence of the sites, and thus can make use of resources from other cloud providers' infrastructure with a simple customer account.

Several directions have been identified for future works:

In this thesis we proposed the IGM algorithm to find the optimal service composition for a complex service. We expect that IGM can be applied in the Mobile Cloud Computing [47, 51]. Mobile Cloud Computing exploits Cloud Computing and mobile broadband to bring online computing powers to mobile terminals. A service composition approach is proposed in Magneto project (CELTIC CP5-012). In this approach, mobile terminals make use of other equipments in its local networks to provide the functionalities and computing power it lacks. The equipments advertise their functionalities and computing powers as services in the network. The service composition mechanism analyses the request of the mobile terminal and its configuration to identify the advertised services suitable for it. For instances, if a user wants to watch a video on his smartphone, the video resolution has to be reduced to adapt to the size of the smartphone's screen, its limited graphical capacity and wireless connection capacity. This functionality can be provided by the transcode service from either his computer or a set-top-box. Thus a complex service is created composed of the media server's broadcast service and the computer's transcode service. IGM can bring optimization to service composition capabilities to better use online services for the benefit of mobile terminals.

In Chapter 4, we do not take into consideration the capacity of the services. In reality, cloud customer (service owner) does not fix their service resources requirements. In function of the demand from end-users, service owner will allocate or release the resources for their services. Therefore, service replicas are different in terms of allocated resources. Replicas which serve more end-users will need more resources than replicas which serve less end-users, thus cost more to service owner. We can find similar problems in "Capacitated Facility Location" topic. However, as we already discussed, Facility Location does not consider the change in the provisioning plan. Therefore, further research is needed to tackle these new constraints.

Our orchestration system (Chapter 5) is a simple resources provisioning mechanism without business support. However, there is a need for business level support. For instance, while scaling up resources, the system does not ensure users' workloads: virtual machines are simply shut down whether it is processing a user's workload or not. Therefore, the system needs enhancements to support business quality.

Since each cloud provider has his own business model, to deploy customers' services on other cloud providers' infrastructure, we need a common understanding of all cloud providers to better serve the customers. The on-going research in Easi-clouds project aims to create an environment where cloud providers agree on common understandings (rules, interfaces, languages, etc.) to allow automatically negotiation, business support and automatic service configuration when deploying services inter-clouds.

Annexe

Annexe 1: Orchestration System API Summary

Message format:

```
<HTTP method> <URL>
<HTTP Headers>
Request: <JSON format>
Response: <JSON format>
```

Header must include “X-Auth-Token" with a valid keystone token. Requests and response data in JSON. All terms within “< ... >" are parameters.

Get information of a complex service:

Method	URL	Description
GET	/<tenant_id>/compound_app	Returns information about the compound application.

Public-IP related commands:

Method	URL	Description
GET	/<tenant_id>/compound_app/floating_ips	Return information about the public IPs allocated to the compound application.
POST	/<tenant_id>/compound_app/floating_ips/new	Allocate a new public IP to the specified server.
POST	/<tenant_id>/compound_app/floating_ips/delete	Deallocate the specified public IP.

Create / modify a complex service:

Method	URL	Description
POST	/<tenant_id>/compound_app	Create a compound application.
POST	/<tenant_id>/compound_app/extend	Extend a compound application: add nodes and links.
POST	/<tenant_id>/compound_app/new_servers	Create new servers within the compound application.
POST	/<tenant_id>/compound_app/delete_servers	Deletes the specified servers within the compound application.
DELETE	/<tenant_id>/compound_app/nodes/<node_name>	Delete the specified node. Servers of the node will be deleted as well as orphan links
DELETE	/<tenant_id>/compound_app/links/<link_name>	Delete the specified link.
DELETE	/<tenant_id>/compound_app/servers	Deletes the all servers. Keep (empty) nodes and links
DELETE	/<tenant_id>/compound_app	Deletes the complex service.

Annexe 2: Orchestration System message example

A - Customer's request

The following is an example of a the customer's request in JSON format. The requests specifies the deployment of a complex services consisting of two nodes: *alpha*, *beta* and a link between these nodes. The node *alpha* contains two virtual machines (*"servers"* tag), which is specified by the *"quantity"* tag. Node *beta* contains only one virtual machine (no *"quantity"* tag is used). All virtual machines are initiated with a same image (determined by the *"image_ref"* tag).

Note that all text after "//" are comments added for human readers. Since JSON does not support comments, these texts must not be included in the real message.

```
POST /dd1e95fa114c4958b7657ff8986e9cb6/compound_app
Content-Type: application/json
Accept: application/json
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

```
{
// Request for creating a new complex service (compound_app)
 "compound_app": {
  // List of required nodes for the complex service
   "nodes": [
     {
      // Node alpha
       "name": "alpha",
       "servers": [
         {
           "name": "alp",
           // URL to retrieve the image, provided by Glance service
           "image_ref": "http://192.168.25.5:9292/v1/images/9bbaf441-d59fbaa9310a",
           "flavor_name": "m1.small",
           // Node alpha contains two virtual machines of the same configuration
           "quantity": 2
         }
       ],
       // Node requirements
       "description": {
        "availability": 95,
        "location": "lab:office"
       }
     },
     {
       // Node beta
       "name": "beta",
       "servers": [
         {
           "name": "bet",
           // URL to retrieve the image, provided by Glance service
           "image_ref": "http://192.168.25.5:9292/v1/images/9bbaf441-d59fbaa9310a",
           "flavor_name": "m1.small",
         }
       ],
   //Node requirements
       "description": {
        "availability": 90
```

```
        }
      }
    ],
    "links": [
      {
        "name": "link1",
        "nodeA_name": "alpha",
        "nodeB_name": "beta",
        // Link requirements
        "description": {
          "size": 0.5 // in GB
        }
      }
    ],
    "extras": {
      "ssh_keypair": {
       // Public key is hiden because it is too long to show here
        "public_key": "<hidden>",
        "name": "key_test"
      }
    }
  }
}
```

B - Server's response

In case of success, the API frontend component will respond with the following message. The link provided in the "Location" tag will point to a log which shows the detailed progress of the deployment. User can get details of the complex service (including process' current state) by sending GET HTTP request to this address. It is the URL used by system API for the GET compound_app message (Annexe 1).

```
HTTP/1.0 201 Created
Date: Fri, 17 Aug 2012 15:03:55 GMT
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Location: http://localhost:8080/dd1e95fa114c4958b7657ff8986e9cb6/compound_app
```

C - Service details

Using the link (“Localtion" tag) which the system responded in Annexe B, the customer can get all the details of the requested complex service deployment. In the following paragraph illustrates a GET command to retrieve the details of the complex service. MOSt responds in JSON format. We note the IP address of the created virtual machines which are the private IP address of each site (“addr" tag). Some of the texts are hidden due to their length.

Customer Request

```
GET /dd1e95fa114c4958b7657ff8986e9cb6/compound_app
Content-Type: application/json
Accept: application/json
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

System Reponse

```
HTTP/1.0 200 OK
Date: Fri, 17 Aug 2012 15:09:12 GMT
Content-Length: 0
Content-Type: text/html; charset=UTF-8

{
    "compound_app": {
        "extras": {
            "ssh_keypair": {
                "name": "key_test",
                "public_key": "<hidden>"
            }
        },
        "links": [
            {
                "description": {
                    "quantity": 0.5
                },
                "extras": {},
                "name": "link1",
                "nodeA_name": "alpha",
                "nodeB_name": "beta"
```

```
        }
    ],
    "nodes": [
        {
            "description": {
                "availability": 95,
                "location": "lab:office"
            },
            "extras": {},
            "name": "alpha",
            "servers": [
                {
                    "OS-DCF:diskConfig": "MANUAL",
                    "OS-EXT-STS:power_state": 1,
                    "OS-EXT-STS:task_state": null,
                    "OS-EXT-STS:vm_state": "active",
                    "accessIPv4": "",
                    "accessIPv6": "",
                    "addresses": {
                        "test-net": [
                            {
                                "addr": "10.1.42.18",
                                "version": 4
                            }
                        ]
                    },
                    "config_drive": "",
                    "created": "2012-08-17T15:03:55Z",
                    "flavor_name": "m1.tiny",
                    "hostId": "f8beaca04cff1212d1fefe880895afa6080fc3cdbbe811ce885b0eeb",
                    "id": "90d280bd-c9a3-4b33-a67c-bc10776dfa5b",
                    "image_ref": "<hidden>",
                    "key_name": "key_test",
                    "links": [
                        {
                            "href": "<hidden>",
                            "rel": "self"
                        },
                        {
```

```
                "href": "<hidden>",
                "rel": "bookmark"
            }
        ],
        "metadata": {
            "app_node": "alpha",
            "meta1": "meta"
        },
        "name": "vm1",
        "status": "ACTIVE",
        "tenant_id": "dd1e95fa114c4958b7657ff8986e9cb6",
        "updated": "2012-08-17T15:04:10Z",
        "user_id": "ea0daa7ab45147558ecfd81636bbd049"
    },
    {
        "OS-DCF:diskConfig": "MANUAL",
        "OS-EXT-STS:power_state": 1,
        "OS-EXT-STS:task_state": null,
        "OS-EXT-STS:vm_state": "active",
        "accessIPv4": "",
        "accessIPv6": "",
        "addresses": {
            "test-net": [
                {
                    "addr": "10.1.42.19",
                    "version": 4
                }
            ]
        },
        "config_drive": "",
        "created": "2012-08-17T15:03:55Z",
        "flavor_name": "m1.tiny",
        "hostId": "f8beaca04cff1212d1fefe880895afa6080fc3cdbbe811ce885b0eeb",
        "id": "e8d062d0-eaab-47b3-bb29-f749ef5bd32d",
        "image_ref": "<hidden>",
        "key_name": "key_test",
        "links": [
            {
                "href": "<hidden>",
```

```
                        "rel": "self"
                    },
                    {
                        "href": "<hidden>",
                        "rel": "bookmark"
                    }
                ],
                "metadata": {
                    "app_node": "alpha"
                },
                "name": "vm2",
                "status": "ACTIVE",
                "tenant_id": "dd1e95fa114c4958b7657ff8986e9cb6",
                "updated": "2012-08-17T15:04:10Z",
                "user_id": "ea0daa7ab45147558ecfd81636bbd049"
            }
        ]
    },
    {
        "description": {
            "availability": 90
        },
        "extras": {},
        "name": "beta",
        "servers": [
            {
                "OS-DCF:diskConfig": "MANUAL",
                "OS-EXT-STS:power_state": 1,
                "OS-EXT-STS:task_state": null,
                "OS-EXT-STS:vm_state": "active",
                "accessIPv4": "",
                "accessIPv6": "",
                "addresses": {
                    "test-net": [
                        {
                            "addr": "10.1.0.99",
                            "version": 4
                        }
                    ]
```

```
                        },
                        "config_drive": "",
                        "created": "2012-08-17T15:04:01Z",
                        "flavor_name": "m1.tiny",
                        "hostId": "bd79dde5a601e16abf36366ac6f75b470c56339985ca6bce0657555e",
                        "id": "442e666e-ba63-4f17-9721-c31384b0f091",
                        "image_ref": "<hidden>",
                        "key_name": "key_test",
                        "links": [
                            {
                                "href": "<hidden>",
                                "rel": "self"
                            },
                            {
                                "href": "<hidden>",
                                "rel": "bookmark"
                            }
                        ],
                        "metadata": {
                            "app_node": "beta"
                        },
                        "name": "vm3",
                        "progress": 0,
                        "status": "ACTIVE",
                        "tenant_id": "dd1e95fa114c4958b7657ff8986e9cb6",
                        "updated": "2012-08-17T15:04:14Z",
                        "user_id": "ea0daa7ab45147558ecfd81636bbd049"
                    }
                ]
            }
        ],
        "os_tenant_id": "dd1e95fa114c4958b7657ff8986e9cb6",
        "unclassified_servers": []
    }
}
```

Bibliography

[1] AWS CloudFormation. *[Online] Available at: http://aws.amazon.com/cloudformation/*.

[2] AWS Import/Export. *[Online] Available at: http://aws.amazon.com/importexport/*.

[3] GNU TAR project. *[Online] http://www.gnu.org/software/tar/*.

[4] Jini. [online]. available: http://www.jini.org/.

[5] Networkx library [online]. avaiable: http://networkx.lanl.gov/.

[6] OpenStack Cloud Software. *Available at : http://www.openstack.org*.

[7] Rackspace cloud. *[Online] http://www.rackspace.com/cloud/*.

[8] Salutation. [online]. available: http://www.salutation.org/.

[9] The Open Group. *[Online] Available at: http://www.opengroup.org*.

[10] Upnp. [online]. available: http://www.upnp.org/.

[11] The Cisco Powered Network Cloud : An Exciting Managed Services Opportunity. *Cisco Whitepaper*, 2009.

[12] Open Data Center Alliance Usage Model. *Open Data Center Alliance. [Online] Available at http://www.opendatacenteralliance.org/ourwork/usagemodels*, 2011.

[13] Top 10 cloud computing providers of 2012. *TechTarget Report. Available at: http://searchcloudcomputing.techtarget.com/photostory/2240149038/Top-10-cloud- providers-of-2012/1/Introduction*, 2012.

[14] Miha Ahronovitz, Dustin Amrhein, Patrick Anderson, Andrew de Andrade, Joe Armstrong, Ezhil Arasan B, James Bartlett, Richard Bruklis, Ken Cameron, Mark Carlson, Reuven Cohen, Tim M. Crawford, Vikas Deolaliker, Pete Downing, Andrew

Easton, Rodrigo Flores, Gaston Fourcade, Thomas Freund, Tom Hanan, Valery Herrington, Babak Hosseinzadeh, Steve Hughes, William Jay Huie, Nguyen Quang Hung, Pam Isom, Shobha J. Rani, Sam Johnston, Ravi Kulkarni, Anil Kunjunny, Edmond Lau, Thomas Lukasik, Bob Marcus, Gary Mazzaferro, Craig McClanahan, Meredith Medley, Walt Melo, Andres Monroy-Hernandez, Ayman Nassar, Dirk Nicol, Lisa Noon, Santosh Padhy, Gilad Parann-Nissany, Greg Pfister, Thomas Plunkett, Ling Qian, Balu Ramachandran, Jason Reed, German Retana, Bhaskar Prasad Rimal, Dave Russell, Matt F. Rutkowski, Clark Sanford, Krishna Sankar, Alfonso Olias Sanz, Mark B. Sigler, Wil Sinclair, Erik Sliman, Patrick Stingley, Phillip Straton, Robert Syputa, Robert J. Taylor, Doug Tidwell, Kris Walker, Kurt Williams, John M Willis, Yutaka Sasaki, Michael Vesace, Eric Windisch, Pavan Yara, and Fred Zappert. Cloud Computing Use Cases A white paper produced by the. *The Open Cloud Manifesto*, 2010.

[15] Artur Andrzejak, Martin Arlitt, and Jerry Rolia. Bounding the Resource Savings of Utility Computing Models. *HP Laboratories Technical Report HPL-2002-339*, 2002.

[16] Julia ANGWIN and Jennifer VALENTINO-DEVRIES. Google's iPhone Tracking. *The Wall Street Journal. [Online] Available at http://online.wsj.com/article/SB10001424052970204880404577225380456599176.html*, 2012.

[17] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. *Report, Electrical Engineering and Computer Sciences, University of California at Berkeley*, 2009.

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

[19] AL Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509, 1999.

[20] S Beco, A Maraschini, and F Pacini. Cloud computing and RESERVOIR project. *Il Nuovo Cimento C*, 32:99–103, 2009.

[21] Rainer Bernnat, Wolfgang Zink, Nicolai Bieber, and Joachim Strach. Standardizing the Cloud A Call to Action. Technical report, Booz & Company, 2012.

[22] Viktors Berstis. Fundamentals of Grid Computing. *IBM Redbook*, pages 1–28, 2002.

[23] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and H. Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd international conference on Virtual execution environments*, pages 169–179. ACM, 2007.

[24] Jeppe Bronsted, Klaus Marius Hansen, and Mads Ingstrup. Service Composition Issues in Pervasive Computing. *IEEE Pervasive Computing*, 9(1):62–70, January 2010.

[25] Carl Brooks. Heroku learns from Amazon EC2 outage. *Search CloudComputing. Available at: http://searchcloudcomputing.techtarget.com/news/1378426/Heroku-learns-from-Amazon-EC2-outage*, 2010.

[26] Rajkumar Buyya, CS Yeo, and Srikumar Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. *High Performance Computing . . .* , 2008.

[27] Zhiping Cai, Fang Liu, Nong Xiao, Qiang Liu, and Zhiying Wang. Virtual Network Embedding For Evolving Networks. *Computational Complexity*, pages 3–7, 2010.

[28] Patrick Callewaert, Paul A. Robinson, and Peter Blatman. Cloud computing: Forecasting change. *Deloitte Report*, 2009.

[29] Nicolas G. Carr. The end of corporate computing. *MIT Sloan Management Review*, 46(3), 2005.

[30] A. Celesti, F. Tusa, M. Villari, and A. Puliafito. Improving Virtual Machine Migration in Federated Cloud Environments. In *Evolving Internet (INTERNET), 2010 Second International Conference on*, pages 61 –67, sept. 2010.

[31] Abhishek Chandra, Weibo Gong, and Prashant Shenoy. Dynamic resource allocation for shared data centers using online measurements. *SIGMETRICS Perform. Eval. Rev.*, 31(1):300–301, 2003.

[32] Anup Chandran, Driss Moutayakine, Terry Ulmer, Manoj Pal, Mahesh Dodani, Gary Craig, Rob Cutlip, Rob Rowe, and Dawn R H Smith. Architecting Portal Solutions. *IBM redbook*, 2003.

[33] Ramnath K. Chellappa and Alok Gupta. Managing computing resources in active intranets. *International Journal of Network Management*, 12(2):117–128, March 2002.

[34] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web services description language (wsdl) version 2.0 part 1: Core language. *W3C Recommendation*, (June):1–103, 2007.

[35] S.Y. Choi and J. Turner. Configuring sessions in programmable networks with capacity constraints. In *IEEE International Conference on Communications, 2003. ICC '03.*, pages 823–829. Ieee, 2003.

[36] S.Y. Choi, J. Turner, and T. Wolf. Configuring sessions in programmable networks. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society*, pages 60–66. Ieee, 2001.

[37] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: Outsourcing Computation without Outsourcing Control. In *Proceedings of the 2009 ACM workshop on Cloud computing security - CCSW '09*, page 85, New York, New York, USA, 2009. ACM Press.

[38] N.M.M.K. Chowdhury, M R Rahman, and R Boutaba. ViNEYard: Virtual Network Embedding Algorithms With Coordinated Node and Link Mapping. *IEEE/ACM Transactions on Networking*, 20(1), 2012.

[39] N.M.M.K. Chowdhury, M.R. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *Proc. IEEE INFOCOM 2009*, pages 783–791. IEEE, 2009.

[40] CompatibleOne-project. CompatibleOne Open Source Cloud Broker Architecture Overview. *CompatibleOne White paper. Available at: http://www.compatibleone.org/bin/view/Discover/Overview*, 10(September):1–10, 2012.

[41] CompatibleOne-project. CompatibleOne Resource Description System (CORDS). *CompatibleOne White paper*, 2012.

[42] Cordys. Cordys Business Operations Platform Architecture. *Cordys Whitepaper*, 2011.

[43] Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON). *RFC 4627*, 2006.

[44] S Crosby, R Doyle, M Gering, and M Gionfriddo. Open Virtualization Format Specification v1.1.0. *DMTF standard No DSP0243*, pages 1–42, 2010.

[45] Doug Davis and Gilbert Pilz. Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol. Technical report, Distributed Management Work Force (DMTF), 2012.

[46] M.D. Dikaiakos, G Pallis, D Katsaros, P Mehra, and A Vakali. Cloud Computing, Distributed Internet Computing for IT and Scientific Research. *IEEE Internet Computing*, 13(5):10–13, June 2009.

[47] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, October 2011.

[48] Ronald P Doyle, Jeffrey S Chase, Omer M Asad, Wei Jin, and Amin M Vahdat. Model-based resource provisioning in a web service utility. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - USITS'03*, Berkeley, CA, USA, 2003. USENIX Association.

[49] P Erdos and A Renyi. On random graphs I. *Publicationes Mathematicae (Debrecen)*, 6:290—-297, 1959.

[50] ETSI. Grid and Cloud Computing Technology : Interoperability and Standardization for the Telecommunications Industry. *European Telecommunications Standards Institute (ETSI) Whitepaper*, 2007.

[51] Niroshinie Fernando, Seng W. Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, January 2013.

[52] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 2000.

[53] Micheal Fitzgerald. Cloud Computing: So You Don't Have to Stand Still. *Online. Avaiable at: http://www.nytimes.com/2008/05/25/technology/25proto.html*, 2008.

[54] Robert W Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, June 1962.

[55] Virtual Galaxy. Virtual Galaxy Cloud building block. *Online. Available at: http://www.vgipl.com/cloud_computing.php?menu=cloud_computing*.

[56] P. Garraghan, P. Townend, and Jie Xu. Real-Time Fault-Tolerance in Federated Cloud Environments. In *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2012 15th IEEE International Symposium on*, pages 118–123, april 2012.

[57] Gartner. Gartner says contrasting views on cloud computing are creating confusion. *Gartner Press releases*, 2008.

[58] Frank Gens. Defining cloud services and cloud computing. *IDC eXchange. [Online]. Available at http://blogs.idc.com/ie/?p=190*, 2008.

[59] D Gmach, J Rolia, L Cherkasova, and A Kemper. Capacity Management and Demand Prediction for Next Generation Data Centers. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 43–50, 2007.

[60] GoGrid. Skydiving through the Clouds. *GoGrid whitepaper*, 1(877).

[61] B Gomolski. Cloud Computing with Ubuntu. *Online. Available at: http://www.ubuntu.com/business/cloud/overview*, 2003.

[62] B Gomolski. Gartner 2003 IT spending and staffing Survey result. *Gartner research, Report. 2003*, 2003.

[63] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. PRESS: PRedictive Elastic ReSource Scaling for cloud systems. In *International Conference on Network and Service Management, CNSM2010*, pages 9–16, 2010.

[64] Amit Goyal and Sara Dadizadeh. A survey on cloud computing. Technical Report December, 2009.

[65] X. Gu, K. Nahrstedt, R.N. Chang, and C. Ward. QoS-assured service composition in managed service overlay networks. In *23rd International Conference on Distributed Computing Systems, 2003. Proceedings.*, volume 23, pages 194–201. Ieee, 2003.

[66] S Guha and S Khuller. Greedy Strikes Back: Improved Facility Location Algorithms,. *Journal of Algorithms*, 31(1):228–248, April 1999.

[67] Andreas Haeberlen. A case for the accountable cloud. *ACM SIGOPS Operating Systems Review*, 44(2):52, April 2010.

[68] Dorit S Hochbaum. Heuristics for the fixed cost median problems. *Mathematical Programming*, 22:148–162, 1982.

[69] Paul Hofmann and D. Woods. Cloud Computing: The Limits of Public Clouds for Business Applications. *IEEE Internet Computing*, pages 90–93, 2010.

[70] Michael Hogan, Fang Liu, Annie Sokol, and Jin Tong. NIST Cloud Computing Standards Roadmap. *National Institute of Standards and Technology (NIST) Special Publication 500-291*, 2011.

[71] Dan Hubbard and Michael Sutton. Top Threats to Cloud Computing V1.0. Technical Report March, Cloud Security Alliance, 2010.

[72] ITNewsWire. Larry Ellison's Brilliant Anti-Cloud Computing Rant. *Online. Avaiable at: http://www.itnewswire.info/2008/09/larry-ellisons-brilliant-anti-cloud.html*, 2008.

[73] K Jain, M Mahdian, and A Saberi. A new greedy approach for facility location problems. *ACM symposium on Theory of computing, Proceedings of the thiry-fourth annual*, 2002.

[74] K. Jeffery and B. Neidecker-Lutz. The Future of Cloud Computing Opportunities for European Cloud Computing Beyond 2010. *European Commission Information Society and Media*, 2010.

[75] Kate Keahey, Patrick Armstrong, John Bresnahan, David LaBissoniere, and Pierre Riteau. Infrastructure outsourcing in multi-cloud environment. *Proceedings of the 2012 workshop on Cloud services, federation, and the 8th open cirrus summit - FederatedClouds '12*, page 33, 2012.

[76] K.M. Khan and Qutaibah Malluhi. Establishing trust in cloud computing. *IT Professional*, 12(5):20–27, 2010.

[77] M Korupolu. Analysis of a Local Search Heuristic for Facility Location Problems. *Journal of Algorithms*, 37(1):146–188, October 2000.

[78] Lee Badger, Robert Bohn, Shilong Chu, Mike Hogan, Fang Liu, Viktor Kaufmann, Jian Mao, John Messina, Kevin Mills, Annie Sokol, Jin Tong, Fred Whiteside, and Dawn Leaf. US Government Cloud Computing Technology Roadmap Volume II Release 1.0. Technical report, National Institute of Standards and Technology (NIST), 2011.

[79] H Liang, O. Kabranov, D. Makrakis, and L. Orozco-Barbosa. Minimal cost design of virtual private networks. *IEEE CCECE2002. Canadian Conference on Electrical*

and Computer Engineering. Conference Proceedings (Cat. No.02CH37373), pages 1610–1615, 2002.

[80] Jin Liang and K. Nahrstedt. Service composition for advanced multimedia applications. *Proceedings of SPIE*, 1(c):228–240, 2005.

[81] Jin Liang and Klara Nahrstedt. Service composition for generic service graphs. *Multimedia Systems*, 11(6):568–581, March 2006.

[82] J.H. Lin and J.S. Vitter. Approximation algorithms for geometric median problems. *Information Processing Letters*, 44(5):245–249, 1992.

[83] M Mahdian, E Markakis, A Saberi, and V Vazirani. A greedy facility location algorithm analyzed using dual fitting. In *Approximation Algorithms for Combinatorial Optimization Problems and 5th International Workshop on Randomization and Approximation, 4th International Workshop on*, volume 50, 2001.

[84] Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. Approximation Algorithms for Metric Facility Location Problems. *SIAM Journal on Computing*, 36(2):411, 2006.

[85] Markets&Markets. Cloud Computing Market: Global Forecast (2010 - 2015). *Markets&Markets report. Available at: http://www.marketsandmarkets.com/Market-Reports/cloud-computing-234.html*, 2010.

[86] Paul Marshall, Henry Tufo, Kate Keahey, David Labissoniere, and Matthew Woitaszek. Architecting a Large-Scale Elastic Environment: Recontextualization and Adaptive Cloud Services for Scientific Computing. In *Proceedings of the 7th International Conference on Software Paradigm Trends (ICSOFT)*, 2012.

[87] Tim Mather, Subra Kumaraswamy, and Shahed Latif.

[88] John Meegan, Gurpreet Singh, Steven Woodward, Salvatore Venticinque, Massimiliano Rak, David Harris, Gerry Murray, Beniamino Di Martino, Yves Le Roux, John McDonald, Ryan Kean, Marlon Edwards, Dave Russell, and George Malekkos. Practical Guide to Cloud Service Level Agreements. *Cloud Standard Consumer Council whitepaper*, pages 1–44, 2012.

[89] Peter Mell and Tim Grance. The NIST definition of cloud computing. *NIST special publication*, 800:145, 2011.

[90] Thijs Metsch. Open Cloud Computing Interface: An Overview. *SNIA Mini summit 2009*, 2009.

[91] Thijs Metsch and Andy Edmonds. Open Cloud Computing Interface - RESTful HTTP Rendering. *Open Grid Forum - OCCI Working group technical report*, 2011.

[92] Thijs Metsch and Andy Edmonds. Open Cloud Computing Interface - Infrastructure. *Open Grid Forum - OCCI Working group technical report*, 2011.

[93] Dejan Milojicic and R. Wolski. Eucalyptus: Delivering a Private Cloud. *Computer*, 44(4):102–104, 2011.

[94] R. Moreno-Vozmediano, R. Montero, and I. Llorente. IaaS Cloud Architecture: From Virtualized Data Centers to Federated Cloud Infrastructures. *Computer*, PP(99), 2012.

[95] K. Muralidharan and P. Gupta. Nimbus: A task aware/context aware mobile computing platform. In *The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*, pages 17–22. IEEE, 2006.

[96] Rajeev Nagaraj. *Threat modeling of para-virtualized environments*. PhD thesis, Wichita State Universit, 2011.

[97] A. Nakao, L. Peterson, and A. Bavier. Constructing end-to-end paths for playing media objects. *Computer Networks*, 38(3):373–389, 2002.

[98] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. Eucalyptus: A Technical Report on an Elastic Utility Computing Archietcture Linking Your Programs to Useful Systems. Technical report, UCSB Computer Science Technical Report Number 2008-10, 2008.

[99] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-source Cloud-computing System. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pages 124–131. IEEE, 2009.

[100] Raft Nyren, Andy Edmonds, Alexander Papaspyrou, and Thijs Metsch. Open Cloud Computing Interface - Core. *Open Grid Forum - OCCI Working group technical report*, 2011.

[101] OASIS. Topology and Orchestration Specification for Cloud Applications - v1.0 - Specification Draft 04. *OASIS Standards Track Work Product*, (August), 2012.

[102] Oracle. Platform-as-a-Service Private Cloud with Oracle Fusion Middleware. *Oracle Whitepaper*, (October), 2009.

[103] Sixto Jr Ortiz. The Problem with cloud computing standradization. *Computer*, (July):13–16, 2011.

[104] G Pallis. Cloud Computing: The New Frontier of Internet Computing. *Internet Computing, IEEE*, 14(5):70, January 2010.

[105] Saurav Pandit. Return of the primal-dual: distributed metric facilitylocation. In *the 28th ACM symposium on Principles of distributed computing*, 2009.

[106] F. Paraiso, N. Haderer, P. Merle, R. Rouvoy, and L. Seinturier. A federated multi-cloud paas infrastructure. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 392–399, june 2012.

[107] A M K Pathan and R Buyya. A taxonomy and survey of content delivery networks. Technical report. GRIDS Laboratory- University of Melbourne- Australia. 2006.

[108] B Raman and R.H. Katz. Load balancing and stability issues in algorithms for service composition. *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, 00(C):1477–1487, 2003.

[109] Ran Raz and Shmuel Safra. A Sub-Constant Error-Probability Low-Degree Test, and a Sub-Constant Error-Probability PCP Characterization of NP. In *Theory of computing, STOC '97, twenty-ninth annual ACM symposium on*, 1997.

[110] Renub. Cloud Computing: SaaS, PaaS, IaaS Market, Mobile Cloud Computing, M&A, Investments, and Future Forecast, Worldwide. *Renub Research*, 2010.

[111] Bart Robertson. Cloud Elasticity - A Real-World Example. 2011.

[112] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan. The RESERVOIR model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):1–11, 2009.

[113] Benny Rochwerger, David Breitgand, and Amir Epstein. Reservoir-when one cloud is not enough. *Computer*, pages 1–7, 2011.

[114] Benny Rochwerger, Alex Galis, Elieze Levy, Juan A Caceres, David Breitgand, Yaron Wolfsthal, Ignacio Martin Llorente, Mark Wusthoff, Rubin S Montero, and Erik Elmroth. RESERVOIR: Management technologies and requirements for next generation Service Oriented Infrastructures. In *Integrated Network Management, 2009. IM '09. IFIP/IEEE International Symposium on*, pages 307–310, 2009.

[115] Florian Rosenberg, Philipp Leitner, Anton Michlmayr, Predrag Celikovic, and Schahram Dustdar. Towards Composition as a Service - A Quality of Service Driven Approach. In *2009 IEEE 25th International Conference on Data Engineering*, pages 1733–1740. Ieee, March 2009.

[116] Joanna Rutkowska and Alexander Tereshkin. Bluepilling the xen hypervisor. Technical report, Black Hat, 2008.

[117] Salesforce.com. Force.com : A Comprehensive Look at the World's Premier Cloud-Computing Platform. *Salesforce.com whitepaper*, 2009.

[118] K Sankar and A. Jones. Cloud Data Management Interface (CDMI) Media Types. *IETF RFC 6208*, pages 1–14, 2011.

[119] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. CloudScale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing - SOCC'11*, New York, NY, USA, 2011. ACM.

[120] David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing - STOC '97*, pages 265–274. ACM Press, 1997.

[121] L. Siegele. Let it rise: A special report on corporate IT. *The Economist*, 2008.

[122] Harjit Singh and D Seehan. Current Trends in Cloud Computing A Survey of Cloud Computing Systems. *International Journal of Electronics and Computer Science Engineering*, 6, 2012.

[123] SNIA. Cloud Data Management Interface v1.0.2. *Storage Networking Industry Association (SNIA) specification*, pages 1–224, 2012.

[124] James Staten, S Yates, and FE Gillett. Is Cloud Computing Ready For The Enterprise? *Forrester Research report*, 2008.

[125] David Stuckey, Stephen Singh, and Skip Scholl. The next generation of cloud computing. Technical report, PricewaterhouseCoopers LLP, Delaware, US, 2011.

[126] Sun. Introduction to Cloud Computing Architecture. *Sun White paper*, (June), 2009.

[127] TechSci. Global Cloud Computing Service Market Outlook 2014. *TechSci report. Available at: http://www.techsciresearch.com/global-cloud-computing-service-market-outlook-2014*, 2010.

[128] TrendMicro. Cloud Security Survey - Global Executive Summary. *Trend Micro Inc. report*, 2011.

[129] J. Veizades, E. Guttman, C. Perkins, and S. Kapman. Service location protocol. *RFC2165*, 1997.

[130] Pavel Vorobiev and Matt Robinson. *Rollbase in Action*. lulu.com, 2011.

[131] M. Wang, B. Li, and Z. Li. sFlow: towards resource-efficient and agile service federation in service overlay networks. In *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, pages 628–635. Ieee, 2004.

[132] Yang Wangli and Wang Huiying. A Service Replic Distribution Scheme Based on Group Structure. In *Proceedings of the 11th Joint Conference on Information Sciences (JCIS)*, pages 1–6, Paris, France, 2008. Atlantis Press.

[133] Wintergreen. Cloud Computing Stack Layers - IaaS PaaS, SaaS Market Strategies, Shares, and Forecasts, Worldwide, 2010-2016. *Wintergreen report. Available at: http://wintergreenresearch.com/reports/cloudSaaS.html*, 2011.

[134] Rafal Wojtczuk and Joanna Rutkowska. Attacking Intel Trusted Execution Technology. Technical report, Invisible Things Lab (ITL), 2009.

[135] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, April 2010.

[136] Qi Zhang, Jin Xiao, E. Gürses, Martin Karsten, and Raouf Boutaba. Dynamic Service Placement in Shared Service Hosting Infrastructures. In *9th International IFIP TC 6 Networking Conference, NETWORKING'2010*, number 1, pages 251–264. Springer, 2010.

[137] Huiyuan Zheng, Jian Yang, and Weiliang Zhao. QoS Analysis and Service Selection for Composite Services. In *2010 IEEE International Conference on Services Computing*, pages 122–129. Ieee, July 2010.

[138] K Zyp and G. Court. A JSON media type for describing the structure and meaning of JSON documents. *IETF Draft.Available at: http://tools.ietf.org/html/draft-zyp-json-schema-03*, 2010.

Publications

International conferences:

1. Khanh-Toan Tran, Nazim Agoulmine, Youssef Iraqi, “Cost-effective complex service mapping in cloud infrastructures", *In Proceedsings of Network Operations and Management Symposium, IEEE NOMS’12*, 2012.
2. Khanh-Toan Tran, Nazim Agoulmine, “Adaptive and Cost-effective Service Placement", *In Proceedings of Global Telecommunications Conference IEEE GLOBECOM’11*, 2011.
3. Hai-Dang Nguyen, Khanh-Toan Tran, Nazim Agoulmine, “Impact of interference on the system performance of Wimax Relay 802.16j with Sectoring", Communications (ICC), *In Proceedings of 2011 IEEE International Conference on Communications, IEEE ICC’11*, 2011.
4. Khanh-Toan Tran, Vamsi Krishna Gondi, Nazim Agoulmine, “Analysis of Client-Based and Network-Based Mobility Protocols", *In Proceedings of 2010 IEEE International Conference on Communications, IEEE ICC’10*, 2010.

Deliverables:

1. A.M. Bosneag et al., “Challenges and Requirements for the Outer Edge Management", *Deliverables, Magneto Project, Celtic CP5-012*, 2009.
2. J. Kielthy et al., “Research Report on Service Centric Network Management and Service Assurance", *Deliverables, Magneto Project, Celtic CP5-012*, 2011.
3. D. Zeghlache et al., “Definition and specification of the infrastructure interfaces and capabilities", *Deliverables, Easi-clouds Project, ITEA 2 - 10014*, 2012. Work in progress.
4. F. Chazal et al., “First Architecture definition and components" , *Deliverables, Easi-clouds Project, ITEA 2 - 10014*, 2012. Work in progress.