POLITECNICO DI MILANO

Scuola di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Generalized Nash Equilibria for the Service Provisioning Problem in Multi-Cloud Systems

Advisor: Prof. Danilo ARDAGNA
Co-Advisor: Prof. Mauro PASSACANTANDO

Master Thesis by:
Ettore TREVISIOL - 770628

Academic Year 2012/2013

This thesis is dedicated to my parents.
For their endless love, support and encouragement.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. Danilo Ardagna, for his excellent guidance, caring, patience and providing me with an excellent atmosphere for doing this thesis. My thanks go also to Prof. Mauro Passacantando for his great help in mathematical issues while developing this work.

I wish to thank my university mates, in particular thanks go to Giorgio Spadaro, Paolo Salvini and Riccardo Sacchi. I am also grateful to Davide Molinari, Anna Savi and the other students of the Software Engineering Laboratory for sharing their knowledge with me during the development of this thesis.

Finally, a particular thank goes to Andrea Fresco and Pier Paolo Cedaro, friends with whom I shared great moments of life.

Ettore

Contents

List of Figures

List of Tables

LIST OF TABLES

Abstract

In recent years the evolution and the widespread adoption of virtualization, service-oriented architectures, autonomic, and utility computing have converged letting a new paradigm to emerge: Cloud Computing. Clouds allow the on-demand delivering of software, hardware, and data as services. Currently the Cloud offer is becoming day by day wider, since all the major IT companies and Service providers, like Microsoft, Google, Amazon, HP, IBM, and VMware have started providing solutions involving this new technological paradigm.

As Cloud-based services are more numerous and dynamic, the development of efficient service provisioning policies becomes increasingly challenging. In this thesis we take the perspective of Software as a Service (SaaS) providers which host their applications at multiple Infrastructure as a Service (IaaS) providers. Each SaaS needs to comply with quality of service requirements, specified in Service Level Agreement (SLA) contracts with the end-users, which determine the revenues and penalties on the basis of the achieved performance level. Each SaaS provider wants to minimize the cost of use of Cloud resources and penalties for requests execution failures. Moreover, SaaS providers compete and bid for the use of infrastructural resources. On the other hand, the IaaSs want to maximize their revenues obtained providing virtualized resources.

In this thesis we model the service provisioning problem as a generalized Nash game. In particular, we develop an efficient distributed algorithm for the run-time allocation of IaaSs resources among competing SaaS providers. We demonstrate the effectiveness of our approach by performing tests considering realistic Cloud scenarios. Numerical results show that our algorithm is scalable and can be used at run-time since it can solve problem instances of maximum size in less than one minute. Compared to other state-of-the-art solutions our model can improve the efficiency of Cloud system evaluated in term of Price of Anarchy up to 100%. Furthermore our analyses point out the SaaS benefits while exploiting multiple IaaS deployment of applications and redistribution of application workloads.

Sommario

Negli ultimi anni l'evoluzione e la diffusa adozione di virtualizzazione, architetture orientate ai servizi, autonomic e utility computing sono confluiti in un nuovo paradigma: il Cloud Computing. Il Cloud Computing ha come obiettivo la fornitura on-demand di software e hardware come risorse accessibili tramite Internet.

Con l'aumento della quantità e della dinamicità dei servizi basati sul Cloud, lo sviluppo di politiche efficienti per la distribuzione delle risorse è diventato sempre più complesso. In questo lavoro di tesi abbiamo studiato il problema dal punto di vista dei fornitori Sofware as a Service (SaaS) che ospitano le loro applicazioni presso molteplici fornitori Infrastructure as a Service (IaaS). Ogni SaaS deve rispettare la qualità del servizio, specificata nei contratti di Service Level Agreement (SLA) con i propri clienti, che determina i ricavi e le penalità sulla base del livello di prestazioni raggiunto. Ogni SaaS vuole minimizzare i costi di utilizzo delle risorse Cloud e delle sanzioni causate dalla violazione dei contratti di SLA. Inoltre, i fornitori SaaS competono fra loro facendo offerte per l'utilizzo delle infrastrutture. Al contrario, gli IaaS vogliono massimizzare i propri introiti ottenuti fornendo risorse virtualizzate.

In questa tesi abbiamo modellato il problema per la fornitura delle risorse come un gioco di Nash generalizzato. In particolare, abbiamo sviluppato un algoritmo distribuito per la gestione a run-time delle risorse degli IaaS fra i SaaS in competizione. Abbiamo dimostrato l'efficacia del nostro approccio compiendo test rappresentativi di scenari di carico reale. I risultati numerici mostrano che l'algoritmo è scalabile e può essere utilizzato a run-time, poiché può risolvere istanze del problema di dimensione massima in meno di un minuto. Rispetto ad altre soluzioni della lettaratura il nostro modello può migliorare l'efficienza in termini di Price of Anarchy del sistema Cloud valutato fino al 100%. Inoltre, le nostre analisi evidenziano i benefici che i SaaS possono ottenere sfruttando il dislocamento delle applicazioni e la distribuzione del traffico su molteplici IaaS.

Chapter 1

Introduction

Cloud Computing has been a dominant IT news topic over the past few years. It is essentially a way for IT companies to deliver software/hardware on-demand as services through the Internet. Cloud Computing applications are generally priced on a subscription model, so end-users may pay a yearly usage fee, for example, rather than the more familiar model of purchasing software licenses. The Cloud-based services are not only restricted to software applications (Software as a Service – SaaS), but could also be the platform for the deployment and execution of applications developed in house (Platform as a Service – PaaS) and the hardware infrastructure (Infrastructure as a Service – IaaS).

In the SaaS paradigm, applications are available over the Web and provide Quality of Service (QoS) guarantees to end-users. The SaaS provider hosts both the application and the data, hence the end-user is able to use and access the service from all over the world. With PaaS, applications are developed and deployed on platforms transparently managed by the Cloud provider. The platform typically includes databases, middleware, and also development tools. In IaaS systems, virtual computer environments are provided as services and servers, storage, and network equipment can be outsourced by customers without the expertise to operate them.

Many companies, e.g., Google, Amazon, and Microsoft are offering Cloud Computing services such as Google's App Engine and Amazon's Elastic Compute Cloud (EC2) or Microsoft Windows Azure. Large data centers provide the infrastructure behind the Cloud and virtualization technology makes Cloud Computing resources more efficient and cost-effective both for providers and customers. Indeed, end-users obtain the benefits of the infrastructure without the need to implement and administer it directly adding or removing capacity almost instantaneously on a "pay-as-you-use" basis. Cloud providers can, on the other hand, maximize the utilization of their

physical resources also obtaining economies of scale.

The development of efficient service provisioning policies is among the major issues in Cloud research. Indeed, modern Clouds operate in a new and dynamic world, characterized by continuous changes in the environment and in the system and performance requirements must be satisfied. Continuous changes occurs without warning and in an unpredictable manner, and are outside the control of the Cloud provider. Therefore, advanced solution need to be developed to manage the Cloud system in a dynamically adaptive way, while continuously providing service and performance guarantees.

The recent evolution of Cloud system and the rapid growth of the Internet have led to a remarkable usage of game-theoretic tools. Problems arising in the ICT industry, such as resource or quality of service allocation problem, pricing, and load shedding, can not always be handled with classical optimization approaches because each player can be affected by the action of all players, not only by his own actions. In this context, game theory models and approaches allow to gain an in-depth analytical understanding of the service provisioning problem.

Game Theory has been successfully applied to diverse problems such as Internet pricing, flow and congestion control, routing, and networking. One of the most widely used "solution concept" in Game Theory is the Nash Equilibrium approach [71]: a set of strategies for the players constitute a Nash Equilibrium if no player can benefit by changing his/her strategy unilaterally or, in other words, every player is playing a *best response* to the strategy choices of his/her opponents.

In this thesis we take the perspective of SaaS providers which host their applications at multiple IaaS providers, thanks to a software layer developed within the MODAClouds project [68] [13]. Each SaaS provider wants to minimize the cost of use of Cloud resources and incurs in penalties in case of requests execution failures. The cost minimization is challenging since online services receive dynamic workloads that fluctuate during the day. Resources have to be allocated flexibly at run-time according to workload fluctuations. Furthermore, each SaaS behaves selfishly and competes with others SaaS for the use of infrastructural resources supplied by the IaaS. Each IaaS, in his turn, wants to maximize the revenues obtained providing the resources.

To capture the behavior of SaaSs and IaaSs in this conflicting situation in which the best choice for one depends on the choices of the others, we recur to the *Generalized Nash Equilibrium* (GNE) concept, which is an extension of the classical Nash equilibrium.

Therefore, the run-time service provisioning problem will be modeled as a *Generalized Nash Equilibrium Problem* (GNEP). We then use Noncooperative Game Theory results to develop an efficient algorithm for the run-time

management and allocation of IaaSs resources to competing SaaSs suitable also for a *fully distributed* implementation. Multiple solutions achieving generalized equilibria will be proposed and evaluated in terms of their efficiency with respect to the *social optimum* of the Cloud. We will demonstrate the effectiveness of our approach by simulation and performing tests on a real prototype environment.

The remainder of the thesis is organized as follows:

- In Chapter 2 we present a general overview on Cloud Computing providing definitions, illustrating the main characteristics and showing the different structures models available. Afterwards, we will explain the state of the art concepts and techniques relative to our work. An analysis and a classification of the literature approaches is given in terms of type of problem, solution found and discipline adopted, according to the approach used: pure optimization or game theory.

- In Chapter 3 the game-theoretic service provisioning problem will be faced. We will start introducing the problem statements and design assumptions. Then an analysis of the game and its properties will be performed. Finally, we will present an algorithm able to find an equilibrium for the resource allocation problem.

- In Chapter 4 we will describe the tools used in this thesis work. A description of the optimization modelers and solvers we adopted will be provided. Furthermore, we will describe the workload injector we have developed to estimate the performance parameters of Cloud applications.

- Chapter 5 will be dedicated to assess the quality of our solution through analyses and experiments. After a description of experiments settings, algorithm scalability and comparison with other two approaches will be performed. Finally, we will analyze the benefits that can be achieved by SaaS when hosting applications on multiple IaaSs.

- In Chapter 6 conclusions will be drawn, underling the achieved results and presenting future research directions.

Chapter 2

State of the art

This chapter presents a general overview on Cloud Computing and explains the state of the art concepts and techniques relative to our work.

After a short introduction on basic notions and advantageous features in Section 2.1, we provide and analyze a definition of Cloud, illustrate the main characteristics and show different structures models in Section 2.2.

In Section 2.3, today's run-time research challenges are presented. A classification of the literature approaches is given in terms of type of problem, solution found and discipline adopted. The papers analyzed in the state of the art are divided according to the approach used: pure optimization or game theory one. This will allow to concentrate and understand the methodology used in this work. Moreover some criteria of evaluation are summarized in final tables.

2.1 Cloud Computing basic concepts

In a world characterized by progress, fast changes and advances, demand for computing power has been increasing over the last half century. Handling workloads of great diversity and enormous scale is necessary in all the most significant fields of today's society, due to the penetration of Information and Communications Technology (ICT) in our daily interactions with the world both at personal and community levels, encompassing business, commerce, education, manufacturing and communication services. With the rapid development of processing and storage technologies, and with the success of the Internet, computing resources have become cheaper, more powerful and more universally available than ever before. In such a setting, dynamic systems are required to provide services and applications that are more competitive, more scalable and more responsive with respect to the classical systems. This

technological trend has enabled the realization of a new computing paradigm called Cloud Computing, in which resources (e.g., CPU, and storage) are provided as general utilities that can be leased and released by users through the Internet in an on-demand fashion.

In a Cloud Computing environment, the traditional role of service provider is divided into two: the infrastructure providers who manage Cloud platforms and lease resources according to a usage-based pricing model, and service-providers, who rent resources from one or many infrastructure providers to serve the end users. The emergence of Cloud Computing has made a tremendous impact on the Information Technology (IT) industry over the past few years, where large companies such as Google [40], Amazon [10] and Microsoft [64] strive to provide more powerful, reliable and cost-efficient Cloud platforms, and business enterprises seek to reshape their business models to gain benefit from this new paradigm. Indeed, Cloud Computing provides several compelling features that make it attractive to business owners.

The new mechanism is increasingly adopted in many areas, such as e-commerce, retail industry and academy for its various advantages:

- *No up-front investments*: Cloud Computing uses a pay-as-you-go pricing model. A service provider does not need to invest in the infrastructure to start gaining benefit from Cloud Computing. It simply rents resources from the Cloud according to its own needs and pays for the usage. As a consequence, cloud model is cost-effective because customers pay for their actual usage without up-front costs.

- *Lowering operating costs*: resources in a Cloud environment can be rapidly allocated and deallocated on demand. Hence, a service provider no longer needs to provision capacities according to the peak load. This ensures huge savings since resources can be released to save on operating costs when service demand is low. In this way, costs are claimed to be reduced and in a public Cloud delivery model capital expenditure is converted to operational expenditure.

- *High scalability and elasticity*: infrastructure providers pool large amount of resources from data centers and make them easily accessible. Scalability is possible via dynamic ("on-demand") provisioning of resources on a fine-grained, self-service basis near real-time, without users having to engineer for peak loads (surge computing). Indeed, a service provider can easily expand its service to large scales in order to handle rapid increase in service demands (e.g., flash-crowd effect).

- *Easy access*: services hosted in the Cloud are generally web-based. Therefore, they are easily accessible through a variety of devices with

Internet connections. These devices not only include desktop and laptop computers, but also cell phones and smart devices. Agility improves with users' ability to re-provision technological infrastructure resources.

- *Reducing business risks and maintenance expenses*: by outsourcing the service infrastructure to the Clouds, a service provider shifts its business risks (such as hardware failures) to infrastructure providers, who often have better expertise than many customers and are better equipped for managing these risks. In this way Cloud Computing guarantees business continuity and disaster recovery. In addition, maintenance of Cloud Computing applications is easier, because they do not need to be installed on each user's computer and can be accessed from different places. Consequently, a service can cut down the hardware maintenance and the staff costs.

However, despite the considerable development and spread of Cloud Computing, it also brings many challenges and new problems in terms of quality of service (QoS), Service Level Agreements (SLA), security, compatibility, interoperability, costs and performance estimation and so on. These issues have been analyzed and studied in the last few years but still a lot of investigation needs to be carefully addressed.

Before presenting the state of the art and discussing the main research challenges, in the next sections we explain what Cloud Computing is, highlighting its key concepts and architectural principles.

2.2 Cloud Computing definition

The origin of the term "Cloud Computing" is obscure as it has never been defined in a unique way and precise circumstance. It appears to derive from the practice of drawing stylized Clouds to denote networks in diagrams of computing and communications systems since the half of the XX century. The word "Cloud" is used as a metaphor for the Internet, based on the standardized use of a Cloud-like shape to denote a network on telephony schematics and later to depict the Internet in computer network diagrams as an abstraction of the underlying infrastructure it represents.

The main idea behind Cloud Computing is not a new one, unlike other technical terms; it is not a new technology, but rather a new operations model that brings together a set of existing technologies to run business in a different way. Indeed, most of the elements used by Cloud Computing, such as virtualization and utility-based pricing, are not new. Instead, Cloud

Computing leverages these existing technologies to meet the technological and economic requirements of today's demand for information technology.

If we consider that with the first available large-scale mainframe in academia and corporations, accessible via thin clients / terminal computers, it became important to find ways to get the greatest return on the investment in them, allowing multiple users to share both the physical access to the computer from multiple terminals as well as to share the CPU time, and eliminating periods of inactivity (time-sharing), we can affirm that the underlying concept of Cloud Computing dates back to the 1950s.

In 1961, John McCarthy was the first to suggest publicly that computer time-sharing technology might result in a future in which computing power and even specific applications could be provided and sold through the public utility business model (like water or electricity). This idea was very popular during the late 1960s, but faded by the mid-1970s, since hardware and telecommunications were not sophisticated and prepared enough for this progressive scheme.

The term "Cloud" has also been used in various contexts such as describing large ATM (Asynchronous Transfer Mode) networks in the 1990s. Telecommunications companies began to offer VPN (Virtual Private Network) services instead of dedicated point-to-point data circuits, with comparable quality of service but at a much lower cost. The Cloud symbol was used to represent the demarcation line between provider's and user's responsibility. This boundary was soon extended to cover servers as well as the network infrastructure.

However, since 2000, the idea has resurfaced in new forms. It was after Google's CEO Eric Schmidt used the word to describe the business model of providing services across the Internet in 2006, that the expression really started to gain popularity. Since then, the term Cloud Computing has been used mainly as a marketing term in a variety of contexts to represent many different ideas. The ubiquitous availability of high-capacity networks, low-cost computers and storage devices as well as the widespread adoption of hardware virtualization, service-oriented architecture, autonomic, and utility computing had led to a tremendous growth in Cloud Computing in various fields of application. This is the reason why Cloud Computing term does not have a standard definition. The lack of general and uniform concept generated not only market hypes, but also a fair amount of skepticism and confusion. For this reason, recently there has been work on standardizing the definition of Cloud Computing. As an example, in [90] the author compared over 20 different definitions from a variety of sources to confirm the following standard definition:

> *Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs.*

Despite the validity of the above definition, in this work we choose to adopt the definition of Cloud Computing provided by the *National Institute of Standards and Technologies* (NIST) [62], as we think it covers all the essential 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.*

2.2.1 Characteristics

The above definition highlights the basic properties of Cloud Computing:

- Ubiquity: the user can totally ignore the location of the hardware infrastructure hosting the required service and make use of the service everywhere and every time he needs through his client application.
- Convenience: the consumer can use a service exploiting remote physical resources, without necessarily buying/acquiring them. As he is just charged for the resources provided according to a pay-per-use mechanism, utility-based pricing lowers service operating cost.
- On-demand activation: a service consumes resources only when it is explicitly activated by the user, otherwise it is considered inactive and the resources needed for its execution can be used for other purposes.

Moreover, the NIST definition also specifies five essential characteristics of Cloud Computing, as stated in [21]:

- On-demand self-service: resources can be allocated or "allocated on-demand", without requiring human interaction with the service's providers. The automated self-organized resource management feature yields high agility that enables service providers to respond quickly to rapid changes in service demand such as the flash crowd effect.

- Broad network access: capabilities are available over the Internet and are accessed through mechanisms that promote use by simple (thin) or complex (thick) client platforms (e.g., any device with Internet connectivity such as mobile phones, laptops, and smart devices). Moreover, to achieve high network performance and localization, many of today's Clouds consist of data center distributed in many locations around the globe. A service provider can easily leverage geo-diversity to achieve maximum service utility.

- Resource pooling: different physical and virtual resources are dynamically assigned and reassigned according to consumer demand and needs; they are pooled by providers to serve multiple resource users and using a multi-tenant model. The customer's control is independent from the exact location of the provided resources but may know a location at a higher level of abstraction (e.g., country, state, or data-center). Such dynamic resource assignment capability provides much flexibility to infrastructure providers for managing their own resource usage and operating costs. Examples of resources include storage, CPUs, memory, network bandwidth, and virtual machines.

- Rapid elasticity: resources can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in; the consumer often perceives unlimited availability of resources that can be purchased in any quantity at any time. Compared to the traditional model that provisions resources according to peak demand, dynamic resource provisioning allows service providers to acquire resources based on the current demand, which can considerably lower the operating cost.

- Measured service: resources usage is always automatically controlled and optimized by leveraging a metering capability at a level of abstraction appropriate to the type of service (e.g., storage, bandwidth, CPU activity time for processing services and so on). This monitoring mechanism provides transparency for both the provider and consumer of the utilized service.

2.2.2 Structure models

This section aims to give a global description of the architectural, business and various operation models of Cloud Computing.

A layered architecture

In a Cloud environment, services owned by multiple providers are co-located in a single data center. The performance and management issues of these services are shared among service providers and the infrastructure provider. The layered architecture of Cloud Computing provides a natural division of responsibilities: the owner of each layer only needs to focus on the specific objectives associated with that layer. However, multi-tenancy also introduces difficulties in understanding and managing the interactions among various stakeholders.

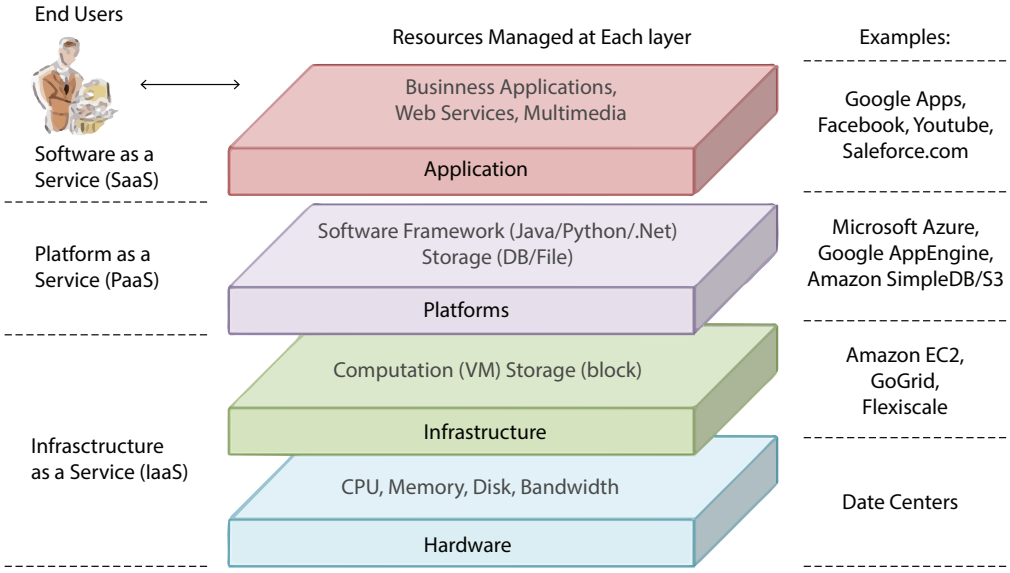

Figure 2.1: Cloud Computing architecture, [106].

Generally speaking, we can identify four layers that compose the architecture of a Cloud Computing environment: the hardware/data center layer, the infrastructure layer, the platform layer and the application layer, as shown in Figure 2.1. This classification allows to understand what each of the layers is composed of, what the intended function of that layer is, and how these layers interact with each other. By simplifying the Cloud Computing concept into layers, it is easier to define the roles within the overall structure and explain where the business fits into the model. A detailed description of each layer follows:

1. *The hardware layer*: this layer is responsible for managing the physical resources of the Cloud, including physical servers, routers, switches,

power suppliers, and cooling systems. In practice, the hardware layer is typically implemented in data centers. A data center usually contains thousands of servers that are organized in racks and interconnected through switches, routers or other fabrics. Typical issues at hardware layer include hardware configuration, fault-tolerance, traffic management, power and cooling resource management. Note that the physical hardware is being sliced into virtual machines (VMs), each having their own small (usually Linux or UNIX based) operating system installed.

2. *The infrastructure layer*: also known as the virtualization layer, the infrastructure layer creates pools of storage and computing resources by partitioning the physical resources using virtualization technologies such as Xen [100], KVM [54] and VMware [92]. These pools of resources are the key to providing elasticity, scalability and flexibility with respect to server architecture. Indeed, virtual machines can be brought online and assigned to a resource pool on-the-fly when the demand on that pool increases, while they can then be destroyed when no longer needed. The ability to provision and delete virtual machines on the fly allows a vendor to provide Infrastructure as a Service (IaaS). As a consequence, instead of purchasing servers or even hosted services, IaaS customers can create and remove virtual machines and network them together at will. Thanks to virtualization technologies, the infrastructure layer offers VMs as a service to end users that have complete control of their environments.

3. *The platform layer*: built on top of the infrastructure layer, the platform layer consists of operating systems and application frameworks, and abstracts the IaaS layer by removing the individual management of virtual machine instances. The purpose of this layer is to minimize the burden of deploying applications directly into VM containers. In fact, at this layer customers do not manage their virtual machines, they merely create their own programs and applications, which are hosted by the platform services they are paying for, within an existing API or programming language. This frees the developers from concerns about environment configuration and infrastructure scaling, but offers limited control.

4. *The application layer*: at the highest level of the hierarchy, the application layer consists of the actual Cloud applications that offer web-based software as a service (SaaS), such as email or CRM (Customer Relationship Management). In this layer, users are truly restricted to only

what the application is and can do, they get only pre-defined functionality and they cannot go much further than that. Indeed, applications are designed for ease of use and GTD (getting things done). Billing can be based on utility or a flat monthly fee. Either way, it is a simple way to get the application functionality you need without incurring the cost of developing that application. Different from traditional applications, Cloud applications can leverage the automatic-scaling feature to achieve better performance, availability and lower operating cost.

We note that the architecture of Cloud Computing is modular, much more than traditional service hosting environments. Each layer is loosely coupled with the layer above and below, allowing each layer to evolve separately. The architectural modularity allows Cloud Computing to support a wide range of application requirements while reducing management and maintenance overhead.

Cloud service models

Cloud Computing adopts a service-driven operating business model, indicating a strong emphasis on service management. In other words, hardware and platform-level resources are provided as services on an on-demand-basis, according to the SLAs negotiated with its customers. Conceptually, every layer of the architecture described in the previous section can be implemented as a service to the layer above. Conversely, every layer can be perceived as a customer of the layer below. However, in practice, Cloud Computing providers offer services that can be grouped into three fundamental categories: infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS), as in Figure 2.2.

- *Infrastructure as a Service*: IaaS refers to on-demand provisioning computers, storage and other infrastructural physical resources, usually in terms of VMs. The Cloud owner who offers IaaS is called an IaaS provider. The consumer is able to deploy and run arbitrary software, which can include operating systems and applications. In this model, the consumer does not manage or control the underlying Cloud infrastructure but he is responsible for patching and maintaining the operating systems, deployed application software, storage, and he possibly has limited control of select networking components (e.g., host firewalls). Cloud providers typically bill IaaS services on a utility computing basis, that is, cost reflects the amount of resources allocated and consumed.

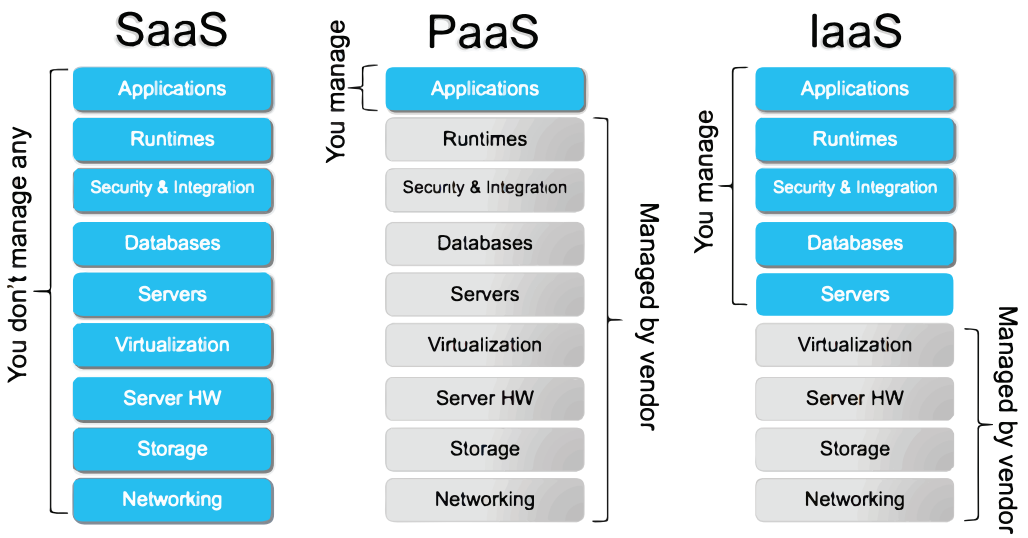

Figure 2.2: Cloud service models.

Examples of IaaS providers include Amazon EC2 [9], Windows Azure Virtual Machines [67], Google Compute Engine [43], GoGrid [39], and Flexiscale [38].

- *Platform as a Service*: PaaS refers to providing platform layer resources, typically including operating system support, database, web server and software development frameworks. These provided capabilities are consumer-created or acquired applications, created using programming languages and tools supported by the provider. The consumer has control over the deployed applications and possibly application hosting environment configurations but cannot manage the underlying Cloud infrastructure. Application developers can develop and run their software solutions on a Cloud platform without the cost and complexity of buying and managing the underlying hardware and software layers. With some PaaS offers, the underlying computer and storage resources scale automatically to match application demand such that Cloud user does not have to allocate resources manually. Examples of PaaS providers include Google App Engine [41], Microsoft Windows Azure Compute [66], and Force.com [78].

- *Software as a Service*: SaaS refers to providing on-demand applications over the Internet. Cloud providers install and operate application software running on a Cloud infrastructure while Cloud users access these

software from various client devices through a thin interface such as a web browser (e.g., web-based email). The consumer does not manage or control the underlying Cloud infrastructure and platform on which the application is running, with the possible exception of limited user-specific application configuration settings. This eliminates the need to install and run the application on the Cloud user's own computers simplifying maintenance and support. The pricing model for SaaS applications is typically a monthly or yearly flat fee per user, so price is scalable and adjustable if users are added or removed at any point. Examples of SaaS providers include Salesforce.com [78], Rackspace [75] and SAP Business ByDesign [79], Google Apps [42], Microsoft Office 365 [65], and Onlive [74].

We observe that PaaS and IaaS providers are often called the infrastructure providers of Cloud providers since, in the current practice, they are often part of the same organization (e.g. Google and Salesforce). However, according to the layered architecture of Cloud Computing, it is entirely possible that a PaaS provider runs its Cloud on top of an IaaS provider's Cloud.

IaaS instance options

In addition to providing the flexibility to easily choose the number, the size and the configuration of the compute instances the customers need for their applications, an IaaS provides customers different purchasing models that give them the flexibility to optimize their costs. For example Amazon EC2, IaaS market leader and reference model of this work, offers three kind of instances: (i) *On-demand* instances which allow the customer to pay a fixed hourly rate with no commitment; (ii) *reserved* instances where the customer pay a low, one-time fee and in turn receive a significant discount on the hourly charge; (iii) *on-spot* instances which enable the customer to bid whatever price he wants for individual instance, providing for even greater savings if his application have flexible start and end times. An accurate description of the three different EC2 instance options follows:

- *On-demand instances*: on-demand instances requires no long-term commitments or upfront payments. Customers can increase or decrease compute capacity depending on the demands of their own applications and only pay the needed rate for the instances they use. An IaaS always strives to have enough on-demand capacity available to meet customers needs, but during periods of very high demand, it is possible that it might not be able to launch specific on-demand instance

types in specific availability zones (i.e., a specific Amazon data center) for short periods of time. On-demand instances are recommended for users that want the low cost and flexibility without any up-front payment or long-term commitment, or applications with short term, spiky, or unpredictable workloads that cannot be interrupted.

- *Reserved instances*: functionally, reserved and on-demand instances perform identically but those reserved let the customer make a low, one-time, upfront payment for an instance, reserve it for a one or three year term, and pay a significantly lower hourly rate for that instance. Customers are assured that their own reserved instance will always be available for the operating system (e.g. Linux/UNIX or Windows) and availability zone in which they purchased it. For applications that have steady state needs, reserved instances can provide high savings compared to using on-demand instances. Reserved instances are usually recommended for applications with predictable usage, applications that require reserved capacity, including disaster recovery and users able to make upfront payments to reduce their total computing costs even further.

- *On-spot instances*: spot instances provide the ability for customers to purchase compute capacity with no upfront commitment and at hourly rates usually lower than the on-demand rate. Spot instances allow you to specify the maximum hourly price that you are willing to pay to run a particular instance type. IaaS sets a spot price for each instance type in each availability zone, which is the price all customers will pay to run a spot instance for that given period. The spot price fluctuates based on supply and demand for instances, but customers will never pay more than the maximum price they have specified. If the spot price moves higher than a customer's maximum price, the customer's instance will be shut down by the IaaS. Other than those differences, spot instances perform exactly the same as on-demand or reserved instances. For the majority of cases, spot instances are recommended for applications that have flexible start and end times, applications that are only feasible at very low compute prices and users with urgent computing needs for large amounts of additional capacity. Due to the nature of on-spot instances, competitions for their acquisition raise between customers. This mechanism can be modeled with game theory approaches as presented in Section 3.2 of this thesis.

Cloud deployment models

There are many issues to consider when moving an enterprise application to the Cloud environment. For example, some service providers are mostly interested in lowering operation cost, while others may prefer high reliability and security. Accordingly, there are different types of Clouds, each with its own benefits and drawbacks:

- *Public Clouds*: a Cloud in which service providers offer resources as services available to the general public or a large industry group and owned by a private organization selling Cloud services (like Amazon AWS, Microsoft and Google); these services are free or offered on a pay-per-use model. Public Clouds offer several key benefits to service providers, including no initial capital investment on infrastructure and shifting of risk to infrastructure providers. However, they lack fine-grained control over data, network and security settings, which restricts their effectiveness in many business scenarios.

- *Private Clouds*: also known as internal Clouds, private Clouds are designed for exclusive use by a single organization. A private Cloud may be hosted internally or externally and managed by the organization or by a third-party represented by external providers. A private Cloud offers the highest degree of control over performance, reliability and security. However, they are often criticized for being similar to traditional proprietary server farms and do not provide benefits such as no up-front capital costs. Moreover, undertaking a private Cloud project requires a significant level and degree of engagement to virtualize the business environment: every one of the steps in the project raises security issues that must be addressed in order to avoid serious vulnerabilities.

- *Community Clouds*: the Cloud infrastructure is shared by several organizations and supports a specific community that has common concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or by a third party and may exist on premise or off premise. The costs are spread over fewer users than a public Cloud (but more than a private Cloud), so only some of the cost savings potential of Cloud Computing are realized.

- *Hybrid Clouds*: a hybrid Cloud is an alternative solution to addressing the limitations of both public and private Clouds. It is a combination of two or more Cloud models (public, private or community), that remain unique entities but are bound together by standardized or

proprietary technology that enables data and application portability (e.g., Cloud bursting for load-balancing between Clouds). In this way companies and individuals are able to obtain degrees of fault tolerance combined with locally immediate usability without dependency on internet connectivity. Hence, hybrid Cloud architecture is flexible and scalable. Compared to public Clouds, they provide tighter control and security over application data, while still facilitating on-demand service expansion and contraction. On the down side, designing a hybrid Cloud requires carefully determining the best split between public and private Cloud components.

For most service providers, selecting the right Cloud model depends on the business scenario. For example, computing-intensive scientific applications are best deployed on public Clouds for cost-effectiveness. Arguably, certain types of Clouds will be more popular that others. In particular, it was predicted that hybrid Clouds will be the dominating deployment model (Figure 2.3) for most organizations.

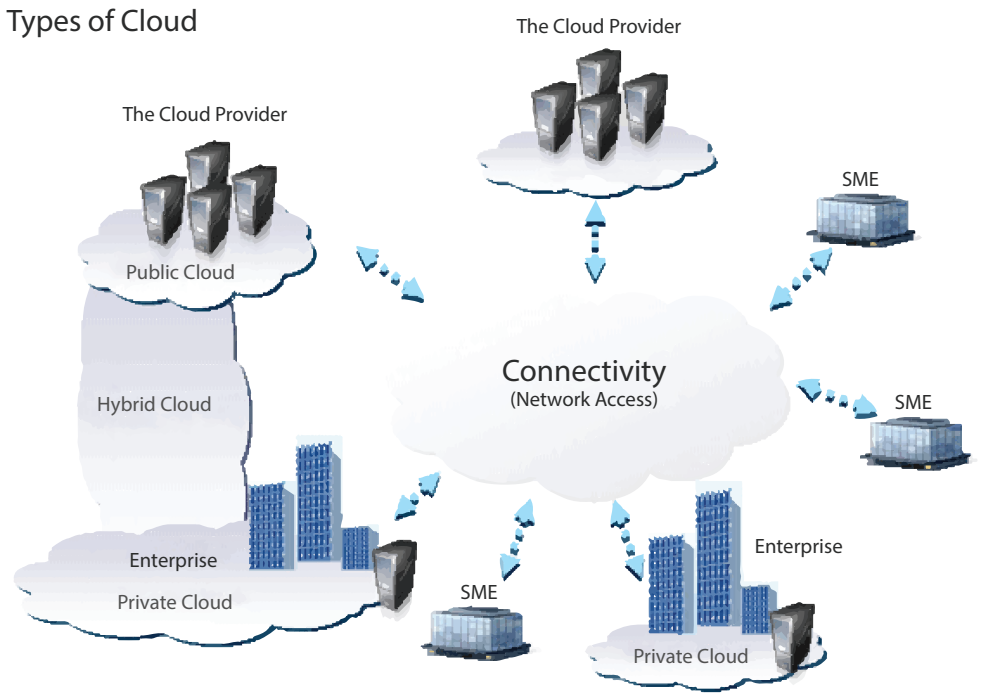

Figure 2.3: Cloud deployment models.

It is fundamental to note that Cloud Computing is still an evolving paradigm. Its definitions, structure, use cases, underlying technologies, issues, risks, and benefits will be refined in a spirited debate by the public and private sectors. The definitions, attributes, and characteristics given in the previous discussion will evolve and change over time. Finally we remark that the Cloud Computing industry represents a large ecosystem of many models, vendors, and market niches. Our description attempts to encompass all of the various Cloud approaches.

2.3 Cloud Computing and run-time research challenges

Despite the significant benefits offered by Cloud Computing, the current technologies are not advanced enough to realize its full potential. As discussed in the previous sections, Cloud Computing is attractive to business owners as it eliminates the requirement for users to plan ahead for provisioning, and allows enterprises to start from the small and increase resources only when there is a rise in service demand. However, despite the fact that IT industry started adopting Cloud Computing, the development of its technology is currently at its infancy, with many issues still to be addressed. Research on Cloud Computing is still at an early stage and, while some aspects of this domain are starting receiving attention from the research communities, new challenges keep emerging from industry applications.

A systematic literature review was already presented in the work of Aleti, Buhnova, Grunske, Koziolek, and Meedeniya, [6]. Since the current state of the art is fragmented over different research communities, multiple system domains, and multiple quality attributes, they have analyzed the results of many research papers on architecture optimization approaches and created a taxonomy which is used to classify the existing research. Their survey aims to provide a basic classification framework integrating the current research results, help the research community in consolidating the existing efforts and point out current trends, gaps, and directions deriving a research agenda for future developments.

Although this cross analysis is very helpful to have a comprehensive overview on the research challenges, we extend the proposed taxonomy according to our objectives. Since we want to understand the different perspectives, point out the types of problem, analyze the solutions found and examine the various disciplines adopted, we present a new and original clas-

sification based on these important categories.

The aim of this section is to provide a better understanding of run-time challenges of Cloud Computing and identify important research directions in this increasingly important area.

Many solutions have been proposed for the management of Cloud service centers at run-time, each seeking to meet application requirements while controlling the underlying infrastructure. Five main problem areas have been considered in resource management policies: 1) application/VM placement, 2) admission control, 3) capacity allocation, 4) load balancing and 5) energy consumption. While each area has often been addressed separately, it is noteworthy that these problem solutions are closely related. For example, the request volume determined for a given application at a given server depends on the capacity allocated to that application on the server. The following discussion aims to figure out how these problems are addressed and to classify them according to theoretical or applied criteria, conforming to the related research developed by the scientific community nowadays.

A first classification of the literature approaches might consider the actor optimizing the use of resources: many proposals take the perspective of the Cloud providers (e.g., IaaS/PaaS) whose goal is to determine the optimal configuration of the underlying infrastructure in order to satisfy incoming requests from the end-users while minimizing some cost metrics. In the opposite perspective the actor involved in resource management optimization is the Cloud end-user which performs cloud resource allocations according to application needs minimizing the cost of use of Cloud resources. In our work we consider the second methodology, in particular we focus on SaaS's side hosting its applications on multiple IaaS.

To provide a comprehensive overview of the current state of the art and classify the research literature, we consider four main dimensions:

1. type of problem,

2. solution found,

3. discipline adopted.

In the next three subsections we will branch out and examine in detail this classification. Thanks to this analysis we can accurately consider the work done in this field in recent years and better understand the demanding and interesting challenges in Cloud Computing.

2.3.1 Problem

The first category we want to consider is related to the problem the approaches aim to solve in the real world. Every approach tries to achieve a certain optimization goal in a specific context. Most of the problems aim to minimize costs, others want to ensure high performance or high availability of the system, some others to simultaneously guarantee these goals. Since the nature and the architecture of a system are concepts difficult to be defined, it is useful to categorize some quantifiable quality attributes as performance, cost, availability, reliability, safety, security or energy consumption.

Furthermore, the set of optimized quality attributes can be aggregated into a single mathematical function or decoupled into conflicting objective: the first one optimizes a single quality attribute only (single-objective optimization, SOO), while the second optimizes multiple quality attributes at once (multi-objective optimization, MOO). Often, for a nontrivial multi-objective optimization problem, there does not exist a single solution that simultaneously optimizes each objective; in that case, the objective functions are said to be conflicting, and there exists a (possibly infinite number of) Pareto optimal solutions. Some approaches encode priority criteria following MOO into a single mathematical function (multi-objective weighted, MOW), others can even use specifically designed functions.

Besides the dimensionality, each problem is characterized even by the quality constraints that represents additional attributes or other system properties. Constraints include structural constraints and performance constraints as arrival rate of applications or available memory bound, containment of resource rental prices, energy costs of the infrastructure use limitation, response time restraint, throughput threshold. In some cases constraints are not present.

2.3.2 Solution

Problems at run-time can be secondly analyze on the solution category. In this way we classify the approaches according to how they achieve the optimization goal and thus describe the main steps of the optimization process. First, solutions type can be centralized or distributed according to the framework and to the interplay between the system factors; alternatively there are hierarchical solutions when the resources are managed according introducing multiple decision points (e.g., clusters and server inside a cluster [25]).

Within each problem, which can be provider selection, application placement, capacity allocation, load-balancing or admission control, it is impor-

tant to identify the degrees of freedom (DoF) exploited by proposed representation of the system under study. Indeed, DoF indicate what changes of the system are considered as variables in the optimization, showing what resolution is best suited to the specific problem in a real environment.

Furthermore, approaches can be characterized according to the solution methods. Firstly, architecture representation classifies the solutions based on the information used to describe the problem structure and configuration: according to the input required, there can be architectural model, UML (Unified Modeling Language), ADL (Architecture Description Language) or optimization models (linear or non-linear). Secondly, concerning with the solution technique, two main categories of optimization strategies can be pointed out: those using exact methods or those guaranteeing approximate solutions. Among exact methods there can be standard methods, branch-and-bound approaches which guarantee a lower bound of the solution or problem-specific methods, while heuristic methods require problem or domain specific information to perform the search and meta-heuristic methods that apply high-level strategies. The latter might exploit for example local search, Evolutionary Algorithms such as Genetic Algorithms, Simulated Annealing or bio-inspired as for example neural networks.

Another characteristic that differentiates the various searches and solution methods is constraint handling that describes the used strategies to handle constraints. More precisely this category distinguishes if they are treated as hard constraints or soft constraints with related penalties.

Finally, solutions are classified according to the time scale used which can range from a daily or hourly scale up to the granularity of minutes, in some cases even seconds.

2.3.3 Discipline

Lastly, the disciplines used to solve these problems at run-time advantage of various disciplines which range from mathematics to computer science. Between the most used we find control theory methods, machine learning that is a branch of artificial intelligence and utility based method consisting of combining performance models and optimization models.

Alternatively, some researchers exploit quantitative models for QoS characteristics analysis such as state based models, Markov Chain and related Markov Decision Processes, Queueing Networks or Logistic Queueing Networks, Stochastic Petri Nets, Process Algebra, and many more.

A main advantage of a control theoretic feedback loop is system stability guarantees. Upon workload changes, these techniques can also accurately

model transient behavior and adjust system configurations within a transitory period, which can be fixed at design time. However, these techniques are typically implemented by local controllers and, hence, only local optimization objectives can be reached. Machine learning techniques are based on live system learning sessions, without a need for analytical models of applications and the underlying infrastructure. An advantage of machine learning techniques is that they accurately capture system behavior without any explicit performance or traffic model and with little built-in system-specific knowledge. However, training sessions tend to extend over several hours, retraining is required for evolving workloads, and existing techniques are often restricted to separately applying actuation mechanisms to a limited set of managed applications. Utility-based approaches have been introduced to optimize the degree of user satisfaction by expressing their goals in terms of user-level performance metrics. Typically, the system is modeled by means of a performance model embedded within an optimization framework. Optimization can provide global optimal solutions or sub-optimal solutions by means of heuristics, depending on the complexity of the optimization model.

Analyzing the state of the art, section 2.3.4, we have divided the previous researches exploiting optimization methods into pure approaches with a single objective function, from game theory ones, with one objective function per player. This distinction enables us to understand how other authors have addressed their researches and make comparison with our work.

2.3.4 State of the art

As presented in the previous section, several approaches have been developed to manage the problems arisen in Cloud Computing. Since it is a new interesting technology, rapidly growing and appealing for industries, there is a multitude of studies in different research fields. Researchers have been concentrating on specific problems, choosing different solution architectures and strategies, using suitable techniques and evaluating the results according to the purpose of the works. The next two sections present some of the most significant works that have carried on in the last few years. As said before pure optimization approaches are divided from works concerning game theory methods.

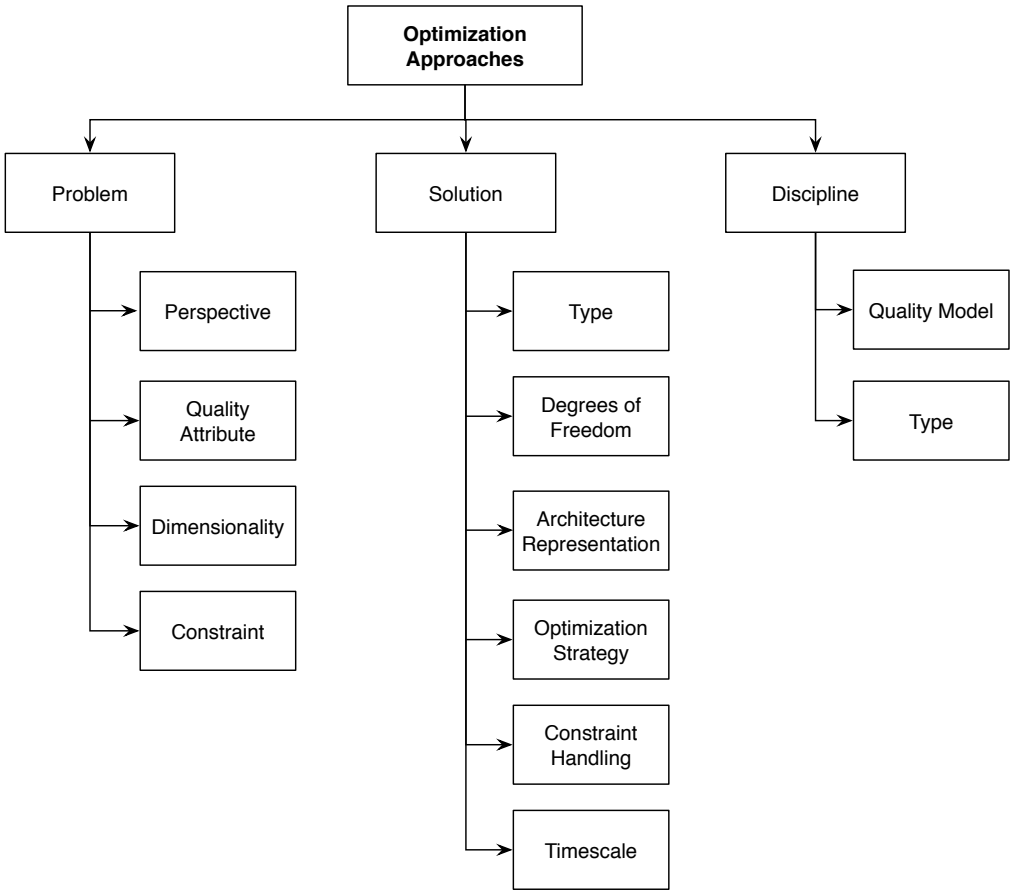

Figure 2.4: Taxonomy for optimization approaches.

Pure optimization approach

There are studies that focus the attention on the QoS, as [57] where the authors use the branch-and-bound approach together with an adjusting recursive procedure to evaluate and maximize the reliability of a computer network in a Cloud Computing environment; the studied algorithm considers budget, time and stochastic capacity constraints. Other studies consider green Cloud Computing, a field of research that concentrates on the sustainable use of energy. In [84] an energy-aware method to evaluate the trade-offs between resource utilization and guaranteed consolidated workload performance is presented, while [61] considers allocation policies and explains how to ensure the user experience and related SLAs while minimizing energy consumption. Note that both works consider heuristics methods.

Other studies deal with the problem of provisioning VMs in the Cloud

with various approaches. In [55] a probabilistic approach aim to test admission control and to find the optimal allocation; the multi-objective weighted function incorporates business rules in terms of trust, eco-efficiency and cost, and it is associated to constraints representing real factors that compromise the Cloud services, including the variable number of users in time and the different patterns of requests. Authors in [105] show the today's limitations for Cloud Computing providers in allocating their VMs with offline mechanisms based on fixed-prices or auctions. Improvements have been demonstrated by implementing a mechanism for online VM provisioning and allocation, called MOVMPA, that aims to maximize the profit of each provider, and by comparing it with current methods. Scheduling techniques are analyzed in [49] where low-priority tasks are opportunistically scheduled in parallel tasks onto underutilized computation resources in the Cloud left by high-priority tasks. A model that manages tasks as ON/OFF Markov chains is presented and a demonstration shows that the optimal solution requires Markov Decision Process solving, which has exponential complexity.

Besides functional and operational approaches, most of the work concentrates the attention on economic aspects such as minimization of costs or, dually, profit maximization. Indeed, a lot of studies, like the one presented in this work, are focused on the conflict between the minimization of costs due to the purchase of resources and SLAs compliance. The work discussed in [103] addresses two challenges: the minimization of the total amount of resources while meeting the end-to-end performance requirements for the applications; using open, closed or semi-closed workloads as input for an adaptive PI controller, an SLA-based control method leads to exact solution of the minimization of the mean round trip time for N-tier web applications with resource partitioning schemes. The same problem, but under a certain financial budget and time constraints, has been investigated in [88] with the use of MapReduce processing. A minimum cost maximum flow algorithm is proposed in [46] to optimally solve the dynamic resource allocation and placement problem, exploiting directed graph and an adaptation of the Bin-Packing algorithm combined with a prediction mechanism. Alternatively, in [44] a force-directed resource assignment (FRA) heuristic is used to optimize the total expected profit for processing, memory requirement and communications resources and it is compared with the capacity constraint relaxation solution that represents an upper-bound. In [24] authors discuss an opportunistic service replication policy that leverages the variability in VM workload and perfomance, as well as the on-demand billing features in the Cloud ensuring response time constraints and maintaining the target system utilization. A cost-aware Cloud provisioning engine is proposed in [81]. The so called Kingfisher system exploits both replication and migration

to dynamically provision capacity and reconfigure applications and it uses an integer linear programming formulation to minimize costs and guarantee throughput constraint. A model for applying revenue management to on-demand IT services have been presented in [59]. The model uses a nonlinear objective function and numerical results are provided to determine the optimal price over different system capacity and multiple classes with different SLAs.

Using Bellman equations and a dynamic bidding policy, in [104], an optimal strategy under a Markov spot price evaluation is found in order to complete deadline jobs with no real-time availability constraints, like in scientific computing. Model's performance are evaluated by a comparison with uniformly distributed spot prices and EC2 spot prices. Another work regarding on-spot bidding is treated in [83]. Authors propose a profit aware dynamic bidding algorithm, which observes the current spot price and selects bid adaptively to maximize the time average profit of the Cloud service broker and minimize its costs in a spot instance market. In [99], to overcome SaaS-IaaS interaction limitation, a cost-effective admission control and scheduling algorithms for SaaS providers are proposed, in order to effectively utilize public Cloud resources to maximize profits by minimizing costs and penalty delays and improving customer satisfaction level.

Regarding autoscaling problem, various approaches have been studied. [29] presents a model-driven engineering approach to optimizing the configuration, energy consumption, and operating costs of Cloud auto-scaling infrastructure to create greener computing environments that reduce emissions resulting form superfluous idle resources. The paper also describes how these models can be transformed into constraint satisfaction problems for configuration and energy consumption optimization in order to create optimal auto-scaling configurations. In [32] authors show SmartScale, an automating scaling framework that uses a combination of vertical (adding more resources to existing VM instances) and horizontal (adding more VM instances) scaling; this method ensures that each application is scaled in order to optimize both resource usage and the reconfiguration cost incurred due to scaling itself. Finally, in [101], an implementation of a system that provides automatic scaling for Internet application is described. Each application is encapsulated in a single VM and the system scales up and down, minimizing costs and energy consumption, maximizing the loads, deciding application placement and load distribution thanks to a color set algorithm.

Game Theory approach

The recent development of Cloud systems and the rapid growth of the Internet have led to a remarkable increase of the research works in the Cloud Computing field. Since Game Theory suits perfectly to the social, economic and strategic structure of Cloud Computing, in the last few years there was a significant improvement in the application of Game Theory tools to this area of study.

Problems arising in the ICT industry, such as resource or quality of service allocation problem, pricing and load shedding, can not be handled with classical optimization approaches. Indeed, in a general complex system, the interrelationships among different users or providers cannot be represented with any tool of pure optimization methods [8]. Therefore, there is a lack of methodologies that can be resolved by Game Theory approach.

In representing Cloud Computing mechanism, interaction across different players is non-negligible: each player can be affected by the actions of all players, not only by his own actions. Game Theory can reproduce perfectly this aspect. In this setting, a natural modeling framework involves seeking an equilibrium, or stable operating point for the system. More precisely, each player seeks to optimize his own goal, which depends on the strategies of the other players upon his own; note that this optimization is performed simultaneously by different players. An equilibrium (in the sense of Nash) is reached when no player can decrease his objective function by changing unilaterally its strategy.

A survey of different modeling and solution concepts of networking games, as well as a number of different applications in telecommunications and wireless networks, based on Game Theory, can be found in [73] [8].

As for the specific problems of Cloud Computing, various theories and various methods have been used to represent, model and manage Cloud services, both at design time and at run-time. Different types of equilibria and game have been considered, according to the problem addressed.

In [37] authors present a methodical in-depth game theory study on price competition, moving progressively from a monopoly market to a duopoly market, and finally to an oligopoly Cloud market. They characterize the nature of non-cooperative competition in a Cloud market with multiple competing Cloud service providers, derive algorithms that represent the influence of resource capacity and operating costs on the solution and they prove the existence of a Nash equilibrium. On the dynamics of the market, a model of competitive equilibrium in e-commerce to solve the problem of pricing and outsourcing can be found in [30]; here the analysis of pricing choices and

decisions to outsource IT capability leads to a representation of the Internet competition and extracts the maximum profit minimizing the costs. In [31] a model of market share of providers with competition is given by some form of "price war" resurfaces in the price-QoS-capacity game, when QoS-capacity relations are determined by explicit queueing formula. Authors in [52] manage the problem of, short or long term, deadlines with axiomatic bargaining approaches. More precisely, they have shown that Nash Bargaining Solution (NBS) ensures proportional fairness whereas Raiffa Bargaining Solution (RBS) can handle real-time task arrivals and dynamics; an asymmetric pricing scheme allows the Cloud service provider to choose different parameters such as user's deadlines and budget requirements for deriving optimal resource allocation. Studies of the maximization of the social welfare as a long-term social utility are discussed in [63]. Considering relevant queueing aspects in a centralized setting, under appropriate convexity assumptions on the operating costs and individual utilities, they established existence and uniqueness of the social optimum. Furthermore, other studies based on a non-cooperative game theory, are presented in [94] where authors employ bidding model to solve the resource allocation problem in virtualized servers with multiple instances competing for resources reaching a unique equilibrium point. Similar discussions can be found in [97]: QoS constrained parallel tasks resource allocation problem is considered. Two practical approximated solutions are proposed, firstly for each single participant problem by linear binary programming, secondly for the final optimal strategy by algorithms that minimize efficiency losses.

The problem of provisioning and reinforcing Cloud Computing infrastructures from security point of view is described in [76] by a minimization of the costs and with constraints that ensure capacity availability. Firstly, they prove that a Nash equilibrium under different formulation is computable in polynomial time, then they derive conditions for reinforcing the infrastructure, and show that higher robustness levels are achieved by limiting the disclosure of information about the infrastructure. [5] considers two simple pricing schemes for selling Cloud instances and studies the trade-off between them. Exploiting Bayes Nash equilibrium they provide theoretical and simulation based evidence suggesting that fixed prices generates a higher expected revenue than the hybrid system.

Regarding VMs provisioning, the problem of live migration is examined in [102], where the concept of "skewness" is introduced to measure the unevenness in the multi-dimensional resource utilization of a server. By minimizing skewness, imposing overload avoidance and green computing, an heuristic is implemented so that it combines different types of workloads and it improves the overall utilization of server resources. Dynamic service placement

is discussed to determine the locations where service applications should be placed so that the hosting cost is minimized while key performance requirements are assured. They further consider the case where multiple service providers compete for resource in a dynamic manner and show that there is a Nash equilibrium solution which is socially optimal.

Cooperative games have been analyzed in [48] and [47]. They propose a game-theoretic solution that ensures mutual benefits so that the Cloud providers are encouraged to form a horizontal dynamic federation platform. Using price-based resource allocation they developed both centralized and distributed algorithms in order to find optimal solutions for these games.

Finally, Generalized Nash games for service provisioning problem have been formulated in [17] and [16] to take the perspective of SaaS providers which host their applications at a IaaS provider. Each SaaS needs to comply with QoS requirements and specified SLAs with the user and at the same time maximize its own revenue while minimizing the cost of use of resource supplied by the IaaS that, on the other end, wants to maximize the revenues obtained providing virtualized resources.

2.3.5 Classification of the state of the art

In what follows, summary tables are presented for each of the elements of the taxonomy (Figure 2.4), in order to classify all the works described above according to the categorization presented at the begin of this section.

Firstly, Table 2.1, summarize the researches that take the perspective of the Cloud provider, or of the Cloud end-user or both.

Perspective	
Cloud provider	[101] [32] [29] [57] [84] [61] [55] [105] [102] [103] [52] [44] [24] [59] [99] [37] [63] [94] [76] [5] [48] [47] [16] [17] [97]
Cloud end-user	[88] [46] [81] [49] [31] [104] [83] [107] [16] [17] [14] [15]

Table 2.1: Problem category: perspective.

Table 2.2 categorizes the various studies carried out on the basis of what the problem center, if it focus on issues of performance or reliability, costs

or energy consumption. In particular, in Table 2.3 we summarize the dimensionality of the problems. If the researches face a single problem with a mono objective function, or they deal with different decision criteria through a multiple objectives function problem. In this case, the reader can see that the methodologies presented in literature adopt SOO rather than MOO.

Quality Attributes	
Performance	[76] [44] [88] [24] [31] [99] [101] [104] [14] [15]
Cost	[59] [104] [83] [63] [81] [46] [30] [47] [55] [37] [14] [15]
Availability	[102] [105] [32] [46] [49] [97]
Reliability	[57] [103]
Energy	[84] [61] [29]

Table 2.2: Problem category: quality attributes.

Dimensionality	
Single-objective optimization	[76] [44] [88] [24] [37] [59] [104] [83] [63] [81] [46] [97] [102] [105] [32] [49] [57] [103] [29] [84] [14] [15]
Multi-objective optimization	[31] [99] [101] [30] [47] [55] [61]

Table 2.3: Problem category: dimensionality.

As for the problem constraints Table 2.4 divided works in 5 sub-category. Also in this case, the vast majority of the reviewed papers presents a specific framework able to manage cost and performance constraints.

Furthermore, it is possible to classify the literature according to the kind of solution proposed. Table 2.5 divides the type of solution, separating the methods that allow to take decision through a central point or through a distributed algorithm. The degrees of freedom sub-category is analyzed in

Constraints	
Cost	[31] [88] [59] [30] [83] [37] [97] [55] [57]
Performance	[99] [44] [88] [37] [46] [102] [49] [57] [29] [14] [15]
Availability	[31] [104] [81] [97] [105] [46]
Throughput	[83] [81] [32]
Memory	[44] [59] [47] [46] [103]

Table 2.4: Problem category: constraints.

Table 2.6. Again, many methodologies allow the user to define its own degrees of freedom whereas the most popular built-in one is the capacity allocation.

Type	
Centralized	[31] [24] [59] [63] [47]
Distributed	[76] [88] [30] [47] [97] [84] [14] [15]
Hierarchical	[25]

Table 2.5: Solution category: type.

Architecture representation is exposed in Table 2.7, and since the most of the works adopt optimization models, their strategy is classified in two categories: exact or meta-heuristic (Table 2.8).

Therefore deeper classification is made in Tables 2.9 and 2.10, respectively for constraints handling and time scale.

Degrees of freedom	
Provider selection	[31] [101] [32] [55] [103] [102]
Application placement	[101] [81] [46] [55] [49] [84]
Capacity allocation	[76] [99] [44] [88] [24] [59] [104] [83] [63] [37] [47] [46] [102] [97] [105] [57] [103] [29] [61] [84] [14] [15]
Load balancing	[101] [14] [15]
Admission control	[55] [99]

Table 2.6: Solution category: degrees of freedom.

Architecture representation	
Architecture model	[101]
Optimization model	[44] [88] [99] [37] [81] [59] [63] [105] [103] [32] [55] [29]

Table 2.7: Solution category: architecture representation.

Optimization strategy	
Exact	[31] [24] [59] [104] [83] [63] [37] [46] [97] [46] [103] [29] [15]
Meta-heuristic	[99] [44] [97] [102] [49] [57] [61] [84] [14]

Table 2.8: Solution category: optimization strategy.

Finally, the last couple of tables (2.11 and 2.12) regroup researches for their discipline categorizing them for typology and quality model adopted. As we can see the most of the works consider utility based approach, while we did not consider any work adopting machine learning discipline since it is not of interest for our run-time purpose.

Constraints handling	
Not presented	[30]
Hard	[37] [49] [14] [15]
Penalty	[59]

Table 2.9: Solution category: constraints handling.

Time scale	
Minute	[99] [14] [15]
Hour	[61] [14] [15]
Day	[37]

Table 2.10: Solution category: time scale.

Type	
Utility based	[31] [44] [88] [30] [63] [37] [99] [97] [102] [14] [15]
Control theory	[104] [55] [46] [103]
Machine Learning	

Table 2.11: Discipline category: type.

Quality Model	
Markov chain	[76] [104] [83]
Queueing network	[31] [14] [15]
State based model	[99]

Table 2.12: Discipline category: quality model.

2.3.6 Criteria for evaluation

In order to assess the quality of the solution methods proposed we can add various evaluation criteria that go beyond the above classification. A method is better than another in some respects, but worse in another way. The trade-off must be evaluated in order to choose the most suitable solution to the run-time problem under consideration.

The main dimensions measured can be the time required to find a solution, or the maximum size of the problem instance that can be solved. These measures depend on practical and physical limitations, specific application under study, industry's aim or research's purpose as well on the chances, tools and resources available.

Another important evaluation criterion is scalability as the ability of the solution method to handle problems of growing size or its ability to enlarge the optimization scope (e.g., adding additional quality metrics or constraints). Last but not least is the quality evaluation of the underlying quality evaluation model, that is the accuracy that can be achieved comparing the QoS metrics evaluated through the model with the real figures measured in the real system.

In what follows a brief analysis of the literature reviewed above according to such criteria is reported. Notice that, although a case study is often presented, information about the model's accuracy or its scalability is hardly provided.

In [84] the study reveals the energy performance trade-offs for consolidations and show that optimal operating points exist. Four simulations are compared, maintaining constant the number of applications but varying disk and CPU utilization, showing that the energy used by the proposed heuristic is about 5.4% more than optimal on an average at 20% tolerance. No information about scalability is reported.

Cost effectiveness and scalability are considered as performance measures for the games in [47]. They analyzed the behavior of each game based on the VM resources supply at steady state. The resource allocation games converged to a steady state after a limited number of iteration whereby no player has a tendency to unilaterally changes its strategy. An evaluation of the scalability of the games in terms of social welfare with different revenue function is also given.

To evaluate the scalability of resource allocation algorithm proposed in [16] the authors considered a very large set of randomly generated instances. The number of SaaS providers has been varied between 10 and 100 while the number of applications between 1000 and 10000. They showed that the problem can be solved in the worst case in less than twenty minutes.

In [32] the researches varied the number of servers in the emulated data center and observed the performance demonstrating that the total cost for their software increases linearly with the number of servers. They also measured that the running time of the algorithm is statistically independent from the number of servers.

Large scale simulation demonstrate that the algorithm presented in [101] is extremely scalable: the decision time remains under 4 seconds for a system with 1000 servers and 10000 applications.

An evaluation of the scalability of the algorithm is also shown in [102] by varying the number of VMs in the simulation between 200 and 1400. The speed of increase is between linear and quadratic, and the decision time is about 1.3 s for a synthetic workload and 0.2 s for a real trace.

A complete scalability study is reported in [46]. The deviation from optimal is shown to be consistently small and tend to vanish as the number of physical machines (PMs) get higher. This means that the algorithm proposed is very close to the optimal for a large number of PMs for a large Cloud provider with many data centers a sector where actually the Bin-Packing algorithm encounters scalability problems and takes longer times to find the optimal solution.

A comparison between algorithms is performed in [99]. When the number of user requests varies from 1000 to 5000 without varying other factors such as deadline and budget, for each algorithm the total profit and average response time has increased.

Finally, in [17] they measured the inefficiency of the two presented algorithms in terms of Price of Anarchy (PoA) and Individual Worst Case (IWC). A very large number of randomly generated instances is considered, the number of SaaS providers varies between 10 and 100, while the number of applications between 100 and 1000, obtaining a PoA lower than 1.01 and 1.42 and a IWC lower than 1.12 and 1.62 with the first and second algorithm, respectively. Furthermore the article focused on the scalability arguing that the algorithms scale linearly with the cardinality of the set of SaaS.

Chapter 3

A game theory service provisioning model

In this chapter we will present a service provisioning model, aiming to minimize SaaS costs and maximize IaaS providers revenues in a distributed Cloud environment, while respecting performance and Service Level Agreement (SLA). A multi-Cloud scenario is considered, we assume that SaaS providers can allocate resources on multiple IaaS that have different service centers around the globe.

In Section 3.1 we start introducing the problem under analysis and our design assumptions. In Section 3.2 the problem is formulated as a Generalized Nash game and its relative properties are detailed in Section 3.3. Finally, in Section 3.4 we will present an algorithm able to find an equilibrium for the resource allocation problem in a finite number of iterations.

3.1 Problem statement and assumptions

We consider SaaS providers using Cloud computing facilities according to the IaaS paradigm to offer multiple transactional Web services (WSs), each service representing a different application.

The hosted WSs can be heterogeneous with respect to resource demands, workload intensities and QoS requirements. The set of IaaS will be indicated as $\mathcal{I}$, the set of WS application offered by the j-th SaaS provider is denoted by $\mathcal{A}_j$, $\mathcal{S}$ will indicate the set of SaaSs while $\mathcal{S}_i$ is the set of SaaS providers running at IaaS $i \in \mathcal{I}$.

A SLA contract, associated with each WS application, is established between the SaaS provider and its end-users. The average response time $E[R_k]$ executing the WS application k has to be less or equal to the given threshold

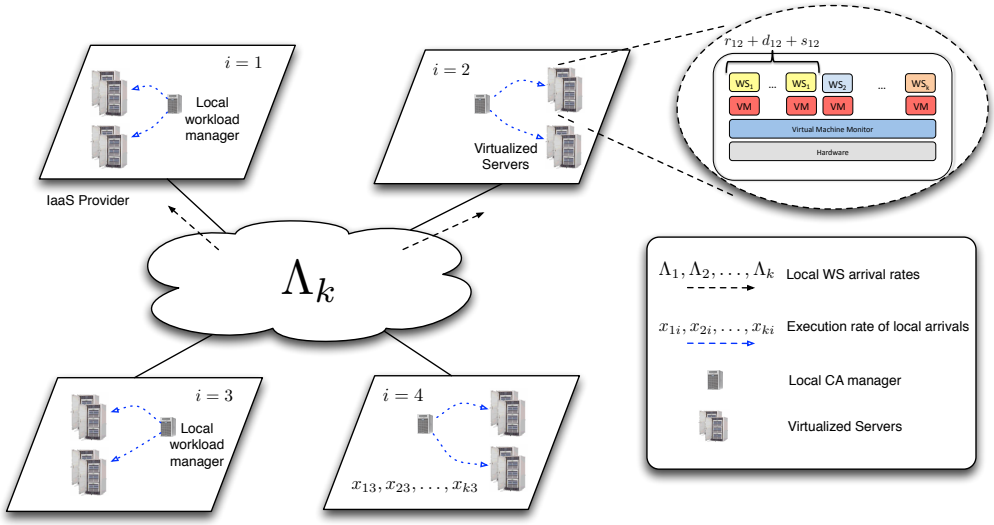

Figure 3.1: Cloud infrastructures.

$\overline{R}_k$, i.e. $R_k < \overline{R}_k$. If the SaaS provider rejects a request it have to pay a penalty ν_k according to the SLA.

Multiple VMs can run in parallel to support the same application. In that case, we suppose that the running VMs are homogeneous in terms of RAM and CPU capacity and the workload is evenly shared among multiple instances (see Figure 3.1), which is common for current Cloud solutions.

Applications are hosted in virtual machines (VMs) which are dynamically instantiated by the IaaS providers up to a maximum of number equal N_i for each IaaS i. We have imposed that each VM is equal to another, providing a service rate μ_{ki} for the application k running at the IaaS i, but this constraint can be easily extended and relaxed. This model wants to implement the possibility for the SaaS providers to sign contracts with one or more IaaS that compete each other in the Cloud market, so we consider the run-time provisioning of resources at the IaaS i. This approach is possible thanks to a software layer developed by the MODAClouds project [68] [13] which allows to migrate at run-time application execution among multiple Cloud providers.

IaaS providers usually charge the use of their resources on an hourly basis. Hence, the SaaS has to face the problem of determining every hour the optimal number of VMs for each WS class in order to minimize costs and penalties, performing resource allocation on the basis of a prediction of future WS workloads. The SaaS needs also an estimate of the future performance of each VM in order to determine application average response time. In

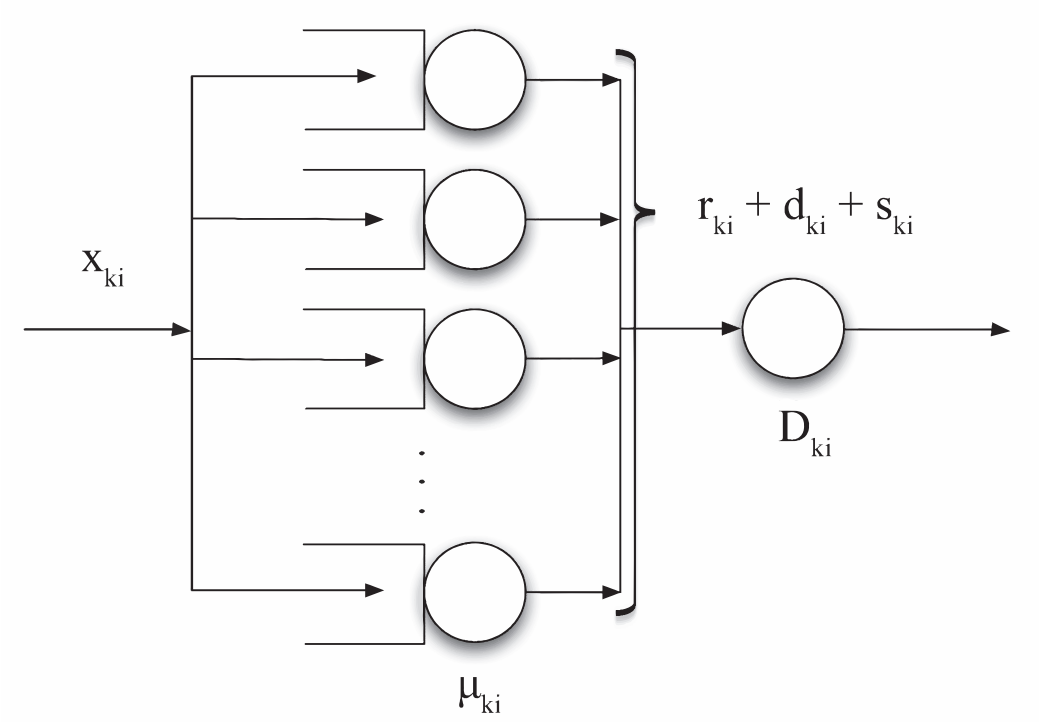

Figure 3.2: System performance model.

the following we model, as a first approximation, each WS class hosted in a VM as an M/G/1 queue (see Figure 3.2) in tandem with a delay center. We assume (as common among Web service containers) that requests are served according to the processor sharing scheduling discipline. The delay center allows to model network delays and/or protocol delays introduced in establishing connections, etc. Performance parameters are also continuously updated at run-time in order to capture transient behavior, VMs network and I/O interference and performance time of the day variability of the Cloud provider.

For the IaaS provider we consider a pricing model similar to Amazon EC2. Given the assumption that the subscript ki means that the application k is hosted at IaaS i, the IaaS provider offers: *reserved* r_{ki} VMs, for which SaaS providers applies for a one-time payment (currently every one or three years) for each instance they want to reserve and *on spot* s_{ki}, for which SaaS providers bid and compete for unused IaaS capacity, or *on demand* d_{ki} VMs that let the SaaS pay for compute capacity by the hour with no long-term commitments.

The VM instances are charged with the on spot cost σ_{ji} for SaaS j hosted

at IaaS i and fluctuates periodically depending on the IaaS provider time of the day energy costs ω_i, on the supply of VMs and demand from SaaS for on spot VMs. On spots costs fluctuate according to the time of the day and on the Cloud site region, and we assume lower than an upper bound σ_{ji}^{Max}. On spot upper bound cost must be strictly lower than δ_i, the on demand VM time unit cost, because no one is willing to pay for a less reliable resource a time unit cost higher than on demand instances which provide a higher availability level, as presented in Section 2.2.2. On spot instances have been traditionally adopted to support batch computing intensive workload during peak periods, but we advocate the use also for traditional transactional services. The reserved instances time-unit cost is equal to ρ_i, according to the SLA contract, and each SaaS j cannot have more than R_{ji} at IaaS i.

Finally, we denote with η_j the maximum fraction of resources allocated as on spot VMs for SaaS provider j, in order to set an upper bound for the $\frac{reserved + on\ demand}{on\ spot}$ ratio to allow a reasonable reliability for every WS application k.

3.2 Generalized Nash game model

In this section we formulate the resource provisioning problem for the Cloud Computing system under study as a Generalized Nash Equilibrium Problem (GNEP) (see Appendix A). The goal of SaaS provider j is to determine every hour the number of reserved r_{ki} VMs, on demand d_{ki}, on spot s_{ki} VMs and the throughput x_{ki} in order to minimize its costs and, at the same time, to satisfy the prediction Λ_k for the arrival rate of the WS application k of SaaS j running at IaaS i to avoid the risk of paying penalties.

Let us denote with μ_{ki} the maximum service rate for the requests of application k, if the workload is evenly shared among the VMs, then the average response time for execution of application k requests is given by:

$$E\left[R_{ki}\right] = D_{ki} + \frac{1}{\mu_{ki} - \left(\frac{x_{ki}}{r_{ki}+d_{ki}+s_{ki}}\right)},$$

where D_{ki} denotes the queuing network delay (see Figure 3.2) and we further assume that the VMs are not saturated (i.e., the equilibrium conditions for the M/G/1 queues hold, $[\,\mu_{ki}\,(r_{ki} + d_{ki} + s_{ki}) - x_{ki}\,] > 0$.

With this settings in mind, the problem that the generic SaaS provider j has to periodically solve becomes:

$$\min_{r_{ki},d_{ki},s_{ki},x_{ki}} \Theta_j = \sum_{k\in\mathcal{A}_j}\sum_{i\in\mathcal{I}_j}(\rho_i\, r_{ki} + \delta_i\, d_{ki} + \sigma_{ji}\, s_{ki}) + \sum_{k\in\mathcal{A}_j}[T\,\nu_k\,(\Lambda_k - X_k)] \quad (3.1)$$

subject to the constraints:

$$D_{ki} + \frac{1}{\mu_{ki} - \frac{x_{ki}}{r_{ki}+d_{ki}+s_{ki}}} \leq \overline{R}_k \qquad \forall\ k \in \mathcal{A}_j, \forall i \in \mathcal{I}_j, \quad (3.2)$$

$$0 \leq s_{ki} \leq \frac{\eta_j}{1-\eta_j}(r_{ki} + d_{ki}) \qquad \forall\ k \in \mathcal{A}_j, \forall i \in \mathcal{I}_j, \quad (3.3)$$

$$x_{ki} < \mu_{ki}\,(r_{ki} + d_{ki} + s_{ki}) \qquad \forall\ k \in \mathcal{A}_j, \forall i \in \mathcal{I}_j, \quad (3.4)$$

$$\sum_{k\in\mathcal{A}_j} r_{ki} \leq R_{ji} \qquad \forall i \in \mathcal{I}_j, \quad (3.5)$$

$$\sum_{k\in\mathcal{A}_i}(r_{ki} + d_{ki} + s_{ki}) \leq N_i \qquad \forall\ i \in \mathcal{I}_j. \quad (3.6)$$

$$\sum_{i\in\mathcal{I}_j} x_{ki} = X_k \qquad \forall k \in \mathcal{A}_j, \quad (3.7)$$

$$\lambda_k \leq X_k \leq \Lambda_k \qquad \forall\ k \in \mathcal{A}_j, \quad (3.8)$$

$$r_{ki}, d_{ki}, s_{ki}, x_{ki} \geq 0 \qquad \forall\ k \in \mathcal{A}_j, \forall i \in \mathcal{I}_j. \quad (3.9)$$

The SaaS goal is to minimize its payoff function (3.1) which includes the fees requested by the IaaSs for instances used and the penalties incurred when requests are discarded.

Constraint (3.2) ensures that the response time is lower than the threshold $\overline{R}_k$ established in the SLA contract. Constraint (3.3) is introduced for fault tolerance reasons, as explained before, and guarantees that the on spot instances are at most a fraction $\eta_j < 1$ of the total capacity allocated for application k at IaaS i. With (3.4) we guarantee that resources are not saturated.

Constraints (3.5) and (3.6) entail that allocated VMs are less or equal to the maximum number offered or guaranteed by the IaaS providers. Constraint (3.5) depends only on applications running at SaaS j, while constraint (3.6) refers to all applications of the set of SaaSs running at IaaS i.

In addition to response time constraints and the virtual machines number due to IaaSs limited capacity there are the throughput constraints. Constraints (3.7) defines the total traffic served by the system as a fraction of the one served by individual IaaSs. Furthermore (3.8) establish a lower bound λ_k for the total throughput needed to satisfy SLA contracts and an upper bound Λ_k which is equal to the total incoming workload.

We remark that, in the formulation of the problem, we have imposed variables r_{ki}, d_{ki} and s_{ki} to be greater or equal to zero (3.9), but not integer, as in reality they are. In fact, requiring variables to be integer makes the solution much more difficult (NP-hard). We therefore decide to deal with continuous variables, actually considering a relaxation of the problem.

On the other side, the IaaS providers' goal (3.10) is to determine the time unit cost σ_{ji} for on spot VM instances for SaaS provider j running applications at IaaS i, in order to maximize their total revenue:

$$\max_{\sigma_{ji}} \Theta_i = \sum_{k \in \mathcal{A}_i} \left[(\rho_i - \omega_i)\ r_{ki} + (\delta_i - \omega_i)\ d_{ki} + (\sigma_{ji} - \omega_i)\ s_{ki} \right] \tag{3.10}$$

with this constraints regarding the on spot cost variables and parameters:

$$\omega_i \leq \sigma_{ji} \qquad \forall\ j \in \mathcal{S}_i, \tag{3.11}$$

$$\sigma_{ji} \leq \sigma_{ji}^{Max} \qquad \forall\ j \in \mathcal{S}_i. \tag{3.12}$$

The on spot instance cost lower bound (3.11) is the energy cost for running a single VM instance for one hour according to the time of the day, and the upper bound (3.12) is a fixed value σ_{ji}^{Max}, lower than the δ_i price for the on demand VMs, established by the SaaS j provider.

In this framework, SaaS providers and the IaaS providers are making decisions at the same time. The decisions of a SaaS depend on those of the others SaaS and the IaaS, while the IaaS objective function depends on SaaS decisions. In this setting, we can not analyze decision in isolation, but we must ask what a SaaS would do, taking into account the decision of the IaaSs and other SaaSs.

To capture the behavior of SaaSs and IaaSs in this conflicting situation (game) in which what a SaaS or the IaaS (the players of the game) does directly affects what others do, we consider the Generalized Nash game, which is broadly used in Game Theory and other fields. We remind the reader that the GNEP differs from the classical Nash Equilibrium Problem since, not only the objective functions of each player (called *payoff functions*) depend upon the strategies chosen by all the other players, but also each player's strategy set may depend on the rival players' strategies, thanks to the limits imposed by the constraint (3.6).

Following the Nash equilibrium concept, SaaS and IaaS providers adopt a strategy such that none of them can improve its revenue by changing its strategy unilaterally (e.g., while the other players keep their strategies unchanged). The service provisioning problem results therefore in a GNEP

where the players are the SaaS providers and the IaaS providers, the strategy variables of SaaS provider j are r_{ki}, d_{ki}, s_{ki} and x_{ki}, for $k \in \mathcal{A}_j$, while the strategy variables of the IaaS i are the costs σ_{ji} for on spot VMs, for all $k \in \mathcal{A}_j$.

For the sake of clarity, the notation adopted here is summarized in Table 3.1.

System Parameters

$\mathcal{S}$	Set of SaaS providers
$\mathcal{I}$	Set of IaaS providers
$\mathcal{S}_i$	Set of SaaS providers j running applications at IaaS i
$\mathcal{I}_j$	Set of IaaS providers supporting SaaS j
$\mathcal{A}$	Set of applications of all the SaaS providers
$\mathcal{A}_j$	Set of applications of the SaaS provider j
$\mathcal{A}_i$	Set of applications running at IaaS i
Λ_k	Prediction of the arrival rate for application k
λ_k	Minimum arrival rate to be guaranteed for application k
μ_{ki}	Maximum service rate for executing class k application at IaaS i
D_{ki}	Queueing delay for executing class k application at IaaS i
$\overline{R}_k$	Application k average response time threshold
ν_k	Penalty for rejecting a single application k request
ρ_i	Time unit cost for reserved VMs for SaaS providers at IaaS i
δ_i	Time unit cost for on demand VMs for SaaS providers at IaaS i
ω_i	VM time unit energy cost for IaaS provider i
η_j	Maximum fraction of total resources allocated as on spot VMs for SaaS provider j
N_i	Maximum number of VMs that can be executed at the IaaS i
R_{ji}	Maximum number of reserved VMs that can be executed for the SaaS j at IaaS i
σ_{ji}^{Max}	Maximum time unit cost offered by IaaS i for on spot VMs instances of SaaS j
T	Control time horizon

SaaS Decision Variables

r_{ki}	Number of reserved VMs used for application k at IaaS i
d_{ki}	Number of on demand VMs used for application k at IaaS i
s_{ki}	Number of on spot VMs used for application k at IaaS i
x_{ki}	Throughput for application k at IaaS i
X_k	Overall throughput for application k

IaaS Decision Variables

σ_{ji}	Time unit cost set by IaaS i to SaaS j for on spot VMs instances

Table 3.1: Parameters and decision variables.

3.3 Game analysis

In this section we study the properties of the game formulated in Section 3.2 in order to demonstrate that for the IaaSs there are dominant strategies, and then we calculate the incentive of all players to change their strategy using a single global function called the potential function.

3.3.1 Dominant strategies for IaaSs

Property 3.3.1. *$\sigma_{ji} = \sigma_{ji}^{Max}$ is a dominant strategy for each IaaS $i \in \mathcal{I}$.*

Proof. In (3.10) we want to maximize the revenue for every IaaS present in the Cloud world. Analyzing the constraints (3.11) and (3.12) we notice that they are only bounding IaaS decision variable σ_{ji}, and since with Θ_i we want to reach the maximum possible gain, we can set $\sigma_{ji} = \sigma_{ji}^{Max}\ \forall i \in \mathcal{I}_j$. In this way we can drop IaaS providers in the game formulation. Furthermore, the SaaS provider payoff function becomes:

$$\min_{r_{ki}, d_{ki}, s_{ki}, x_{ki}} \Theta_j = \sum_{k \in \mathcal{A}_j} \sum_{i \in \mathcal{I}_j} \left[\rho_i \, r_{ki} + \delta_i \, d_{ki} + \sigma_{ji}^{Max} \, s_{ki} + T \, \nu_k \, (\Lambda_k - X_k) \right] .$$

□

3.3.2 Game potential

The model resulting from the optimization problems described in the previous sections is a Generalized Nash Equilibrium Problem with joint constraints, where the players are the SaaS providers whose strategies are r_{ki}, d_{ki}, s_{ki}, x_{ki}.

Since every SaaS objective function is independent from that of all other SaaS we can conclude that this game is a potential game [69] where the potential function is shown in the following result.

Property 3.3.2. *The function*

$$\Pi \left(x_{ki},\, r_{ki},\, d_{ki},\, s_{ki} \right) = \sum_{j \in \mathcal{S}} \sum_{k \in \mathcal{A}_j} \sum_{i \in \mathcal{I}_j} \left(\rho_i \, r_{ki} + \delta_i \, d_{ki} + \sigma_{ji}^{Max} \, s_{ki} - T \, \nu_k \, x_{ki} \right)$$

is a potential for the game.

Proof. In order to calculate the game potential, we have to consider only the objective functions part that use decision variables. Given the simplification obtained in Section 3.3.1, in the calculation of the potential, we can only consider Θ_j,

$$\Theta_j = \sum_{k \in \mathcal{A}_j} \sum_{i \in \mathcal{I}_j} \left(\rho_i \, r_{ki} + \delta_i \, d_{ki} + \sigma_{si}^{Max} \, s_{ki} \right) + \sum_{k \in \mathcal{A}_j} \left[T \, \nu_k \left(\Lambda_k - X_k \right) \right]$$

where we replace X_k with the $\sum_{i \in \mathcal{I}_j} x_{ki}$ as stated in (3.7),

$$\begin{aligned} \Theta_j &= \sum_{k \in \mathcal{A}_j} \sum_{i \in \mathcal{I}_j} \left(\rho_i \, r_{ki} + \delta_i \, d_{ki} + \sigma_{si}^{Max} \, s_{ki} \right) + \sum_{k \in \mathcal{A}_j} \left(T \, \nu_k \, \Lambda_k - \sum_{i \in \mathcal{I}_j} x_{ki} \right) = \\ &= \sum_{k \in \mathcal{A}_j} \sum_{i \in \mathcal{I}_j} \left(\rho_i \, r_{ki} + \delta_i \, d_{ki} + \sigma_{si}^{Max} \, s_{ki} - T \, \nu_k \, x_{ki} \right) + \sum_{k \in \mathcal{A}_j} \left(T \, \nu_k \, \Lambda_k \right) , \end{aligned}$$

then dropping $T \, \nu_k \, \Lambda_k$, since it is a constant, we get $\Pi = \sum_{j \in \mathcal{S}} \Theta_j$.

□

3.3.3 Analysis of constraints

We want to demonstrate that the only non-linear constraint present in the formulation can be rewritten as linear, that allow us to use linear solver that give us a solution faster than those for non-linear formulations.

Property 3.3.3. *In any equilibrium condition for every SaaS j, constraint (3.2) holds as equality.*

$$D_{ki} + \frac{1}{\mu_{ki} - \frac{x_{ki}}{r_{ki} + d_{ki} + s_{ki}}} = \overline{R}_k \qquad \forall \; k \in \mathcal{A}_j, \forall i \in \mathcal{I}_j \tag{3.13}$$

Proof. Assume, by contradiction to have a solution $\left(\overline{r}_{ki}, \, \overline{d}_{ki}, \, \overline{s}_{ki}, \, \overline{x}_{ki} \right)$ such that

$$D_{ki} + \frac{1}{\mu_{ki} - \frac{\overline{x}_{ki}}{\overline{r}_{ki} + \overline{d}_{ki} + \overline{s}_{ki}}} < \overline{R}_k \qquad \forall \; k \in \mathcal{A}_j, \forall i \in \mathcal{I}_j,$$

if we substitute $\overline{r}_{ki}$ with $\tilde{r}_{ki} = \overline{r}_{ki} - \epsilon$, $\overline{d}_{ki}$ with $\tilde{d}_{ki} = \overline{d}_{ki} - \epsilon$ and $\overline{s}_{ki}$ with $\tilde{s}_{ki} = \overline{s}_{ki} - \epsilon$, keeping fixed the other decision variable $\overline{x}_{ki}$ and parameters, we will obtain a new solution vector $\left(\tilde{r}_{ki},\, \tilde{d}_{ki},\, \tilde{s}_{ki},\, \overline{x}_{ki}\right)$, such that $\Theta_j\left(\tilde{r}_{ki},\, \tilde{d}_{ki},\, \tilde{s}_{ki},\, \overline{x}_{ki}\right) > \Theta_j\left(\overline{r}_{ki},\, \overline{d}_{ki},\, \overline{s}_{ki},\, \overline{x}_{ki}\right)$, that is impossible.

□

Furthermore, after some algebra, constraint (3.13) can be written as

$$[1 - p_{ki}\,\mu_{ki}]\,(r_{ki} + d_{ki} + s_{ki}) + p_{ki}\,x_{ki} = 0 \qquad \forall\; k \in \mathcal{A}_j, \forall i \in \mathcal{I}_j,$$

where $p_{ki} = \left(\overline{R}_k - D_{ki}\right)$.

Indeed, from (3.2), since p_{ki}, $[\mu_{ki}\,(r_{ki} + d_{ki} + s_{ki}) - x_{ki}]$ and $(r_{ki} + d_{ki} + s_{ki})$ are positive:

$$\begin{aligned}
\frac{1}{\frac{\mu_{ki}(r_{ki}+d_{ki}+s_{ki})-x_{ki}}{r_{ki}+d_{ki}+s_{ki}}} &= p_{ki} \\
\frac{r_{ki} + d_{ki} + s_{ki}}{\mu_{ki}\,(r_{ki} + d_{ki} + s_{ki}) - x_{ki}} &= p_{ki} \\
r_{ki} + d_{ki} + s_{ki} &= p_{ki}\,[\mu_{ki}\,(r_{ki} + d_{ki} + s_{ki}) - x_{ki}] \\
r_{ki} + s_{ki} &= p_{ki}\,\mu_{ki}\,(r_{ki} + d_{ki} + s_{ki}) - p_{ki}\,x_{ki} \\
[1 - p_{ki}\,\mu_{ki}]\,(r_{ki} + d_{ki} + s_{ki}) + p_{ki}\,x_{ki} &= 0 \qquad \forall k \in \mathcal{A}_j, \forall i \in \mathcal{I}_j.
\end{aligned}$$

Thanks to the previous result we can drop x_{ki} variables and the constraint family (3.4) and we get:

$$x_{ki} = \left[\frac{p_{ki}\mu_{ki} - 1}{p_{ki}}\right](r_{ki} + d_{ki} + s_{ki}) = \left[\mu_{ki} - \frac{1}{p_{ki}}\right](r_{ki} + d_{ki} + s_{ki}) \quad (3.14)$$

Now we can substitute the result obtained in (3.14) in the constraint (3.4) under analysis, having as result:

$$\left[\mu_{ki} - \frac{1}{p_{ki}}\right](r_{ki} + d_{ki} + s_{ki}) < \mu_k\,(r_{ki} + d_{ki} + s_{ki})$$

that simplified as

$$\left[\mu_{ki} - \frac{1}{p_{ki}}\right] < \mu_{ki}$$

and since p_{ki} is positive as shown above, become always true, hence constraint (3.4) then can be eliminated from the game formulation.

Furthermore constraints (3.7) and (3.8) can be rewritten as

$$\lambda_k \leq \sum_{i \in \mathcal{I}_j} x_{ki} \leq \Lambda_k \qquad \forall k \in \mathcal{A}_j.$$

At this point of our discussion, it is fundamental to examine the relation between penalties and costs of VMs. To better understand the SaaS point of view we will show the trade-off between the payments due to the rejection or to the use of the VMs.

Assume that SaaS j is using only on demand VMs at a certain hour of the day. Reminding that Λ_k is the prediction of the arrival rate for application k, from the algebra discussed in the proof of Property (3.3.3), we can write:

$$(1 - p_{ki}\mu_{ki})\, d_{ki} + p_{ki}\Lambda_k = 0$$

from which we obtain an expression for the number of reserved VMs at site i to execute the entire class k (i.e., Λ_k requests):

$$d_{ki} = \frac{p_{ki}}{p_{ki}\mu_{ki} - 1}\Lambda_k.$$

Multiplying the last expression for the time unit cost for on demand VMs δ_i, we have an upper bound of the total cost per hour:

$$\Psi_i = \delta_i \frac{p_{ki}\Lambda_k}{p_{ki}\mu_{ki} - 1}.$$

Finally, the unit cost per hour to execute the request k is:

$$\Psi_i^{unit} = \frac{\delta_i p_{ki}}{p_{ki}\mu_{ki} - 1}.$$

In practice, rejecting requests is not convenient for the SaaS. Indeed, SaaS providers choose this solution only if there are no possibilities to pay for the use of VMs. This is the reason why we need to add to our problem a fundamental hypothesis which is satisfied in the current practice and that links penalties with VMs costs:

$$\max_i \frac{\delta_i p_{ki}}{p_{ki}\mu_{ki} - 1} << \nu_k \tag{3.15}$$

where ν_k indicates the penalty to be paid for every rejected request. Therefore, the above assumption expresses the tradeoff between the two different

payment values: in the optimization process the SaaS will use all the VMs available and avoid rejecting as far as possible.

Taking into account the above discussion on the relation between penalty and VMs costs, we can consider another property of the game that links constraints (3.4) and (3.5), having for simplicity:

$$c_{ki} = \mu_{ki} - \frac{1}{\overline{R}_k - D_{ki}} = \mu_{ki} - \frac{1}{p_{ki}} .$$

.

Property 3.3.4. *Given an equilibrium of the GNEP. If it exists an application $k \in \mathcal{A}_j$ such that*

$$\sum_{i \in \mathcal{I}_j} c_{ki} \left(r_{ki} + d_{ki} + s_{ki} \right) < \Lambda_k$$

SaaS variables r_{ki} satisfy constraint (3.5) as equality:

$$\sum_{k \in \mathcal{A}_j} r_{ki} = R_{ji} \quad \forall i \in \mathcal{I}_j .$$

Proof. Suppose, by contradiction, to have

$$\sum_{k \in \mathcal{A}_j} r_{ki} < R_{ji} \quad \forall i \in \mathcal{I}_j ,$$

which means that the SaaS j is using less reserved instances than the available at each IaaS i for it. This happens only when the SaaS cannot wish to improve its payoff function Θ_j. Given Property 3.15, this reflects in:

$$\sum_{i \in \mathcal{I}_j} c_{ki} \left(r_{ki} + s_{ki} \right) < \Lambda_k \quad \forall k \in \mathcal{A}_j ,$$

that is impossible. □

3.3.4 Game model reformulation

Thanks to the simplifications, analysis and the dominant strategy studied in the previous sections we can reformulate the game model, coming to a more compact formulation as below.

$$\min_{r_{ki},\, d_{ki},\, s_{ki}} \Theta_j = \sum_{k \in \mathcal{A}_j} \sum_{i \in \mathcal{I}_j} \rho_i \, r_{ki} + \delta_i \, d_{ki} + \sigma_{ji}^{Max} \, s_{ki} -$$

$$- \sum_{k \in \mathcal{A}_j} \sum_{i \in \mathcal{I}_j} \left[\mu_{ki} - \frac{1}{\left(\overline{R}_k - D_{ki}\right)} \right] \left(r_{ki} + d_{ki} + s_{ki}\right) \, T \, \nu_k \tag{3.16}$$

subject to:

$$0 \leq s_{ki} \leq \frac{\eta_j}{1 - \eta_j} \left(r_{ki} + d_{ki}\right) \qquad \forall \; k \in \mathcal{A}_j, \forall i \in \mathcal{I}_j \tag{3.17}$$

$$\lambda_k \leq \sum_{i \in \mathcal{I}_j} \left[\mu_{ki} - \frac{1}{\left(\overline{R}_k - D_{ki}\right)} \right] \left(r_{ki} + d_{ki} + s_{ki}\right) \qquad \forall \; k \in \mathcal{A}_j \tag{3.18}$$

$$\sum_{i \in \mathcal{I}_j} \left[\mu_{ki} - \frac{1}{\left(\overline{R}_k - D_{ki}\right)} \right] \left(r_{ki} + d_{ki} + s_{ki}\right) \leq \Lambda_k \qquad \forall \; k \in \mathcal{A}_j \tag{3.19}$$

$$\sum_{k \in \mathcal{A}_j} r_{ki} \leq R_{ji} \qquad \forall i \in \mathcal{I}_j \tag{3.20}$$

$$\sum_{k \in \mathcal{A}_i} \left(r_{ki} + d_{ki} + s_{ki}\right) \leq N_i \qquad \forall \; i \in \mathcal{I}_j \tag{3.21}$$

$$r_{ki}, d_{ki}, s_{ki}, x_{ki} \geq 0 \qquad \forall \; k \in \mathcal{A}_j, \forall i \in \mathcal{I}_j \tag{3.22}$$

3.4 A distributed algorithm for identifying Generalized Nash Equilibria

We now present an algorithm (see Figure 3.3) for finding a Generalized Nash Equilibrium. In order to simplify the notation, we denote by

$$y_j := (r_{ki}, d_{ki}, s_{ki})_{k \in \mathcal{A}_j, i \in \mathcal{I}_j}$$

the strategy vector of SaaS j, by $g_j(y_j) \leq 0$ the constraints of SaaS j which are independent from strategies of other players, and by

$$h_i(y) := \sum_{j \in \mathcal{S}_i} \sum_{k \in \mathcal{A}_j} (r_{ki} + d_{ki} + s_{ki}) - N_i \leq 0 \qquad \forall \; i \in \mathcal{I},$$

the shared constraints.

At first iteration each SaaS provider j finds the optimal solution $\widetilde{y}_j$ of his relaxed problem where the joint constraints are removed. If the solution $\widetilde{y}$

satisfies the shared constraints, then it is a social equilibrium. Otherwise, for each IaaS provider i such that the corresponding shared constraint h_i is violated by $\widetilde{y}$, the VMs of IaaS i are shared proportionally among SaaS providers according to the solution $\widetilde{y}$, i.e., we set the number:

$$N_{ij} = \frac{\sum_{k \in \mathcal{A}_j} (\widetilde{r}_{ki} + \widetilde{d}_{ki} + \widetilde{s}_{ki})}{\sum_{j \in \mathcal{S}_i} \sum_{k \in \mathcal{A}_j} (\widetilde{r}_{ki} + \widetilde{d}_{ki} + \widetilde{s}_{ki})} N_i$$

of the resources of IaaS i which will be used by SaaS j in the successive iterations. Next, each SaaS provider solves his own problem with these new individual constraints. If all the shared constraints are satisfied in the new solution, then it is a Generalized Nash Equilibrium, otherwise we compute N_{ij} for each violated shared constraint and we add to each SaaS provider the corresponding constraints as before.

Theorem 3.4.1. *The algorithm in Figure 3.3 finds a Generalized Nash Equilibrium after a finite number of iterations.*

Proof. We note that the set $\mathcal{I}_L$ represents the set of IaaS providers with limited resources with respect to the requests of SaaS providers at previous iterations, while $\mathcal{I}_N$ represents the set of IaaS providers with limited resources with respect to the requests of SaaS providers at the current iteration. The algorithm stops a finite number of iterations because $\mathcal{I}_L$ is a subset of $\mathcal{I}$ with increasing cardinality, thus after a finite number of iterations the set $\mathcal{I}_N$, which is a subset of $\mathcal{I} \setminus \mathcal{I}_L$, becomes empty.

Two cases can occur: either the algorithms stops at the first iteration with $\mathcal{I}_L = \emptyset$ or it stops after several iterations with $\mathcal{I}_L \neq \emptyset$.

In the first case we prove that the found solution $\widetilde{y}$ is a social equilibrium. In fact, each SaaS provider j finds individually the optimal solution $\widetilde{y}_j$ of the relaxed problem without the shared constraints, thus there are optimal KKT multipliers $\widetilde{\beta}_j$ such that the following system holds for all $j \in \mathcal{S}$:

$$\begin{cases} \nabla \Theta_j(\widetilde{y}_j) + \widetilde{\beta}_j^T \nabla g_j(\widetilde{y}_s) = 0 \\ \widetilde{\beta}_j^T g_j(\widetilde{y}_j) = 0 \\ \widetilde{\beta}_j \geq 0 \\ g_j(\widetilde{y}_j) \leq 0. \end{cases}$$

Since $\widetilde{y}$ also satisfies all the shared constraints, i.e. $h_i(\widetilde{y}) \leq 0$ for all $i \in \mathcal{I}$, then $(\widetilde{y}, \widetilde{\beta}, \widetilde{\gamma})$, with $\widetilde{\gamma}_{ij} = 0$ for all $i \in \mathcal{I}$ and $j \in \mathcal{S}_i$, solves the KKT system

0. (Inizialization) Set $\mathcal{I}_L = \emptyset$.

1. (Solution of relaxed subproblems)

for all $j \in \mathcal{S}$ **do**

Find an optimal solution $\widetilde{y}_j$ of $\begin{cases} \min\ \Theta_j(y_j) \\ g_j(y_j) \leq 0 \end{cases}$

end

2. (Stopping criterion)

Compute the set $\mathcal{I}_N := \{i \in \mathcal{I} \setminus \mathcal{I}_L : h_i(\widetilde{y}) > 0\}$.

if $\mathcal{I}_N = \emptyset$ **then** STOP.

else compute

$$N_{ij} = \frac{\sum_{k \in \mathcal{A}_j} (\widetilde{r}_{ki} + \widetilde{d}_{ki} + \widetilde{s}_{ki})}{\sum_{j \in \mathcal{S}_i} \sum_{k \in \mathcal{A}_j} (\widetilde{r}_{ki} + \widetilde{d}_{ki} + \widetilde{s}_{ki})} N_i \qquad \forall\ i \in \mathcal{I}_N,\ \forall\ j \in \mathcal{S}_i,$$

and set $\mathcal{I}_L := \mathcal{I}_L \cup \mathcal{I}_N$.

3. (Solution of new subproblems)

for all $j \in \mathcal{S}$ **do**

Find an optimal solution $\widetilde{y}_j$ of

$$\begin{cases} \min\ \Theta_j(y_j) \\ g_j(y_j) \leq 0 \\ h_{ij}(y_j) := \sum\limits_{k \in \mathcal{A}_j} (r_{ki} + d_{ki} + s_{ki}) - N_{ij} = 0, \quad \forall\ i \in \mathcal{I}_L \cap \mathcal{I}_j \end{cases} \tag{3.23}$$

end

and go to step 2.

Figure 3.3: Algorithm for finding Generalized Nash Equilibria.

associated to the social problem

$$\begin{cases} \min \sum_{j \in \mathcal{S}} \Theta(y_j) \\ g_j(y_j) \leq 0, \qquad \forall\ j \in \mathcal{S}, \\ h_i(y) \leq 0, \qquad \forall\ i \in \mathcal{I}. \end{cases}$$

Since the social problem is linear, it follows that $\widetilde{y}$ is a social equilibrium.

We now consider the second case: the algorithm stops after several iterations with $\mathcal{I}_L \neq \emptyset$. For each SaaS j we know that $\widetilde{y}_j$ is an optimal solution of the problem (3.23), thus there are KKT multipliers $(\widetilde{\beta}_j, \widetilde{\gamma}_{ij})$ such that the following system holds:

$$\begin{cases} \nabla\Theta_j(\widetilde{y}_j) + \widetilde{\beta}_j^T\, \nabla g_j(\widetilde{y}_j) + \sum_{i \in \mathcal{I}_L \cap \mathcal{I}_j} \widetilde{\gamma}_{ij} \nabla h_{ij}(\widetilde{y}_j) = 0 \\ \widetilde{\beta}_j^T\, g_j(\widetilde{y}_j) = 0 \\ h_{ij}(\widetilde{y}_j) = 0, \qquad \forall\ i \in \mathcal{I}_L \cap \mathcal{I}_j \\ \widetilde{\beta}_j \geq 0 \\ g_j(\widetilde{y}_j) \leq 0. \end{cases}$$

We note that multipliers $\widetilde{\gamma}_{ij}$ are nonnegative because if there was a $\widetilde{\gamma}_{ij} < 0$, then the SaaS j could reduce his objective function if the constraint on the VMs of the IaaS i was

$$\sum_{k \in A_j} (r_{ki} + d_{ki} + s_{ki}) < N_{ij},$$

but this is impossible because in a previous iteration the SaaS j had already requested the IaaS i more than N_{ij} VMs.

For each $i \in \mathcal{I}_L$ we have

$$h_{ij}(\widetilde{y}_j) = 0, \qquad \forall\ j \in \mathcal{S}_i,$$

which is equivalent to

$$\sum_{k \in \mathcal{A}_j} (\widetilde{r}_{ki} + \widetilde{d}_{ki} + \widetilde{s}_{ki}) = N_{ij}, \qquad \forall\ j \in \mathcal{S}_i,$$

thus

$$h_i(\widetilde{y}) = \sum_{j \in \mathcal{S}_i} \sum_{k \in A_j} (r_{ki} + d_{ki} + s_{ki}) - N_i = \sum_{j \in \mathcal{S}_i} N_{ij} - N_i = 0,$$

i.e., the shared constraint h_i is active at $\widetilde{y}$.

Moreover, we note that $\nabla h_{ij}(y_j) = \nabla_{y_j} h_i(y)$. If we define $\widetilde{\gamma}_{ij} = 0$ for all $i \in (\mathcal{I} \setminus \mathcal{I}_L) \cap \mathcal{I}_j$, then $\widetilde{y}_j$ satisfies the following system:

$$\begin{cases} \nabla\Theta_j(\widetilde{y}_j) + \widetilde{\beta}_j^T \nabla g_j(\widetilde{y}_j) + \sum\limits_{i \in \mathcal{I}_j} \widetilde{\gamma}_{ij} \nabla_{y_j} h_i(\widetilde{y}) = 0 \\ \widetilde{\beta}_j^T \, g_s(\widetilde{y}_j) = 0, \\ \widetilde{\gamma}_{ij} \, h_i(\widetilde{y}) = 0, \qquad \forall \; i \in \mathcal{I}_j, \\ \widetilde{\beta}_j \geq 0 \\ g_j(\widetilde{y}_j) \leq 0 \\ \widetilde{\gamma}_{ij} \geq 0, \qquad \forall \; i \in \mathcal{I}_j \\ h_i(\widetilde{y}) \leq 0, \qquad \forall \; i \in \mathcal{I}_j. \end{cases}$$

This means that $\widetilde{y}_j$ is the optimal solution of the problem:

$$\begin{cases} \min \;\; \Theta_j(y_j) \\ g_j(y_j) \leq 0 \\ h_i(y_j, \widetilde{y}_{-j}) \leq 0, \qquad \forall \; i \in \mathcal{I}_j, \end{cases}$$

that is $\widetilde{y}_j$ is the best reply of player j to the strategies $\widetilde{y}_{-j}$ of the other players. i.e., $\widetilde{y}$ is a Generalized Nash Equilibrium.

□

Chapter 4

Tools

In this thesis we have developed a software that uses optimization modelers and solvers such as AMPL and CPLEX for solving the problems described in the previous chapter, which will be introduced in Section 4.1 and 4.2.

Afterward Section 4.3 and Section 4.4 describe some tools we have used to estimate the performance parameters of Cloud applications with a workload injector we developed (outlined in Section 4.5) able to automate testing in a Cloud platform.

Finally, in Section 4.6 we present a set of tools we developed to perform analytical analyses in order to validate our algorithm and perform experiments that are the focus of the next chapter.

4.1 AMPL

AMPL is the acronym for "A Modeling Language for Mathematical Programming" [11], developed by Robert Fourer, David Gay and Brian Kernighan at Bell Laboratories: it is an algebraic modeling language for linear and nonlinear optimization problems, in discrete or continuous variables. AMPL does not solve problems directly; instead it communicates with another mathematical software called solver which is responsible for finding the problem's best solution.

One of the main advantages of AMPL is its syntax: it is very similar to the mathematical optimization problems notation making it very readable and simple to understand. It is available for the most important 32- and 64-bit platforms including Linux, Mac OS X and Windows and there is a free student version limited only in the number of variables and constraints.

This tool is ideal for model development and we used it to model the sites resource allocation and the algorithms presented in this thesis.

There are several solvers available on the market, each developed to solve a single or more problems classes. Concerning linear programming the most used is CPLEX which can solve continuous, integer and network problems. Regarding nonlinear programming there is not a unique solver which can solve every problem type because of the complexity of this field: there are CPLEX and MOSEK for quadratic problems, MOSEK and SNOPT for convex ones, SNOPT and KNITRO for continuous and finally MINLP for integer ones.

4.2 CPLEX

For our model we used CPLEX 12.2.0.0 [50] as solver, since its capabilities cover all our requirements. CPLEX is designed to solve linear programs of AMPL, as well as the integer programs. Integer programs may be pure (all integer variables) or mixed (some integer and some continuous variables); integer variables may be binary (taking values 0 and 1 only) or may have more general lower and upper bounds. For the network linear programs, CPLEX also incorporates an especially fast network optimization algorithm.

The barrier algorithmic option to CPLEX, though originally designed to handle linear programs, also allows the solution of a special class of nonlinear problems, namely, quadratic programs (QPs). However, CPLEX does not solve general (non-QP) nonlinear programs.

4.2.1 CPLEX algorithms for continuos optimization

For problems with linear constraints, CPLEX employs either a simplex method or a barrier method to solve the problem. Four distinct methods of optimization are incorporated in the CPLEX package:

- a primal simplex algorithm that first finds a solution feasible in the constraints, then iterates toward optimality;
- a dual simplex algorithm that first finds a solution satisfying the optimality conditions, then iterates toward feasibility;
- a network primal simplex algorithm that uses logic and data structures tailored to the class of pure network linear programs;
- a primal-dual barrier (or interior-point) algorithm that simultaneously iterates toward feasibility and optimality, optionally followed by a primal or dual crossover routine that produces a basic optimal solution.

For problems with quadratic constraints, only the barrier method is used and there is no crossover algorithm.

The simplex algorithm maintains a subset of basic variables (or, a basis) equal in size to the number of constraints. A basic solution is obtained by solving for the basic variables, when the remaining nonbasic variables are fixed at appropriate bounds.

Each iteration of the algorithm picks a new basic variable from the nonbasic ones, steps to a new basic solution, and drops some basic variable at a bound. The coefficients of the variables form a constraint matrix, and the coefficients of the basic variables form a nonsingular square submatrix called the basis matrix. At each iteration, the simplex algorithm must solve certain linear systems involving the basis matrix. For this purpose CPLEX maintains a factorization of the basis matrix, which is updated during most iterations, and is occasionally recomputed. The sparsity of a matrix is the proportion of its elements that are not zero. The constraint matrix, basis matrix and factorization are said to be relatively sparse or dense according to their proportion of nonzeros. Most linear programs of practical interest have many zeros in all the relevant matrices, and the larger ones tend also to be the sparsest.

The amount of RAM memory required by CPLEX grows with the size of the linear program, which is a function of the numbers of variables and constraints and the sparsity of the coefficient matrix. The factorization of the basis matrix also requires memory allocation; the amount is problem-specific, depending on the sparsity of the factorization. When memory is limited, CPLEX automatically makes adjustments that reduce its requirements, but that usually also reduce its optimization speed. For these reasons after a first version of the model implementation we had to compute a code optimization due to an high memory usage and consequent slowdown of the optimization speed for large instances. Optimization has focused in particular on parameters loading, set definitions and the subscripts of the variables in the model.

4.3 SPECweb 2005

SPECweb2005 [86] is the Standard Performance Evaluation Corporation (SPEC) benchmark for evaluating the performance of World Wide Web Servers. SPECweb2005 continues the SPEC tradition of giving Web users the most objective and representative benchmark for measuring a system ability to act as a web server. The SPECweb2005 benchmark includes many sophisticated features like:

- measures simultaneous user sessions,
- relevant dynamic content: ASPX, JSP, and PHP implementations included,
- page requests through parallel HTTP connections,
- multiple, standardized workloads: Banking (HTTPS), E-commerce (HTTP and HTTPS), and Support (HTTP),
- file accesses closely matching today's real-world web server access patterns.

The SPEC defined a set of rules that has to be fulfilled in order to obtain fair results that can be actually published. The SPECweb tool consists of 3 main components:

- web applications,
- back-end simulator,
- load generator.

A set of 3 web applications emulate different type of websites. These applications have to be deployed on the system that will be tested. Real world web applications usually make use of a back-end system in order to reply to user requests. The back-end simulator (BeSim) is used to supply web applications with data requested by the user. This part of the tool has been introduced with the intent to better emulate the environment on which the tested system is going to work. BeSim has been developed in order to avoid bottleneck on this part of the system and efficiently test the web server. The load generator is used to produce the requests that the web server is going to serve. It is composed of 2 logical components:

- Prime Client,
- Client.

The Prime Client is a tool that coordinates many clients, that may be running on many machines, in order to produce the desired workload and gather statistics. The Client is a piece of software that can be run many times concurrently on the same machine or on remote systems, the main job of clients is to generate the workload as specified by the prime client.

The system that will actually be tested is the one on which the web server runs. In order to test entirely the system SPECweb provides 3 web applications called banking, e-commerce and support:

- Banking application emulates a web site of an online bank which let the user log into his account, transfer founds and make payments. The main characteristic of this application is that most of the connections are SSL based so the web server has to decode messages before processing them. This task generates heavy load on the CPU of the system.

- E-commerce application is used to emulate a big variety of web applications, it consists of an e-commerce web site that let the user search, customize and buy products. The requests to this applications are a mix of http and https. E-commerce application is not designed to stress a single component of the system.

- Support application is used to stress the network interface of the system. It emulate a website that hosts files of different size (to a maximum of 40MB).

Clients generate random pattern of requests in order to emulate different users according to a Markov chain model which probabilities has been calculated using logs of real websites.

SPECweb tests are characterized by `SIMULTANEOUS_SESSIONS` parameter, that represents the number of users connected to the web server during the entire duration of the analysis. This value cannot be modified during test execution and consequently we cannot deploy a test characterized by a fluctuating number of users. For this reason we developed a tool based on Apache JMeter [12], described below, with the aim of replacing the limited SPECweb client.

4.4 JMeter

Apache JMeter is an open source and cross platform software, a 100% pure Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions. It is part of the Apache Jakarta Project. It can be used to simulate a heavy load on a server to test its strength or to analyze overall performance under different load types. JMeter can be used to make a graphical analysis of performance or to test a server behavior under heavy concurrent load. Apache JMeter can load and test the performance of many different server types: Web - HTTP, HTTPS, SOAP, Database via JDBC, LDAP, JMS, Mail - POP3(S) and IMAP(S). Apache JMeter main features are: full multithreaded framework, careful GUI design

allows faster operation and more precise timings, caching and offline analysis/replaying of test results, highly extensible. As far as web-services and remote services are concerned, JMeter looks like a browser (or rather, multiple browsers); however JMeter does not perform all the actions supported by browsers. In particular, JMeter does not execute the Javascript found in HTML pages. Nor does it render the HTML pages as a browser does.

JMeter tests

JMeter tests are specified in `.jmx` files, each one representing a Test Plan. The Test Plan object has a checkbox called "Functional Testing". If selected, it will cause JMeter to record the data returned from the server for each sample.

The main elements of a test plan are:

- A ThreadGroup, the starting point of any test plan. It sets: the number of threads, the ramp-up period, the number of times to execute the test.
- Samplers, tell JMeter to send requests to a server and wait for a response. They are processed in the order they appear in the tree. JMeter samplers include: FTP Request, HTTP Request, JDBC Request, Java object request, LDAP Request, SOAP/XML-RPC Request, WebService (SOAP) Request.
- Logic Controllers, allow to customize the logic that JMeter uses to decide when to send requests. Logic Controllers can change the order of requests coming from their child elements.
- Listeners, provide access to the information JMeter gathers about the test cases while JMeter runs. The Graph Results listener plots the response times on a graph.
- Timers, by default, a JMeter thread sends requests without pausing between each request. The timer will cause JMeter to delay a certain amount of time before each sampler which is in its scope.

We decided to utilize JMeter for our purpose because of its usability and flexibility to create our own test plan which emulate workloads and users behavior of SPECweb tests.

4.5 SPECweb deployment in the Cloud

The main problem of using SPECweb to test the Cloud is that this tool has been developed to target a different kind of platforms. It requires some initializations steps that could not be easily performed in a dynamic environment such as the Cloud, it cannot handle dynamic number of simultaneous sessions, and there is no management or monitoring tools for Cloud infrastructures. Therefore we report both SPECweb and JMeter functionalities in order to better understand how we merge them into a new tool able to manage them in the Cloud.

4.5.1 SPECweb tests

Behaviour

SPECweb load injectors work in a closed model environment. Simultaneous sessions are started with a fixed number of users and continuously send dynamic pages with an average interval of 10 seconds (`THINK_TIME`) between them, then they access to a new page, or leave the system.

Every test starts with his own configuration file, with fixed parameters, and stable behavior during the workloads execution time. The diagram in Figure 4.1 reflects the different iteration/phases for the Banking, E-commerce, and Support workloads.

- Phase A: ramp-up period is the time period across which the load generating threads are started. This phase is designed to ramp up user activity rather than beginning the benchmark run with an immediate and full-load spike in requests and sends at least one request per user thread.

- Phase B: the warm-up period is intended to be a time during which the server can initialize its cache prior to the actual measurement interval. At the end of the warm-up period, all results are cleared from the load generator, and recording starts a new one. Accordingly, any errors reported prior to the beginning of the run period will not be reflected in the final results for this benchmark run.

- Phase C: the run period is the interval during which benchmark results are recorded. The results of all requests sent and responses received during this interval will be recorded in the final benchmark results. During this phase the system power data is collected for later retrieval into the results files by the harness.

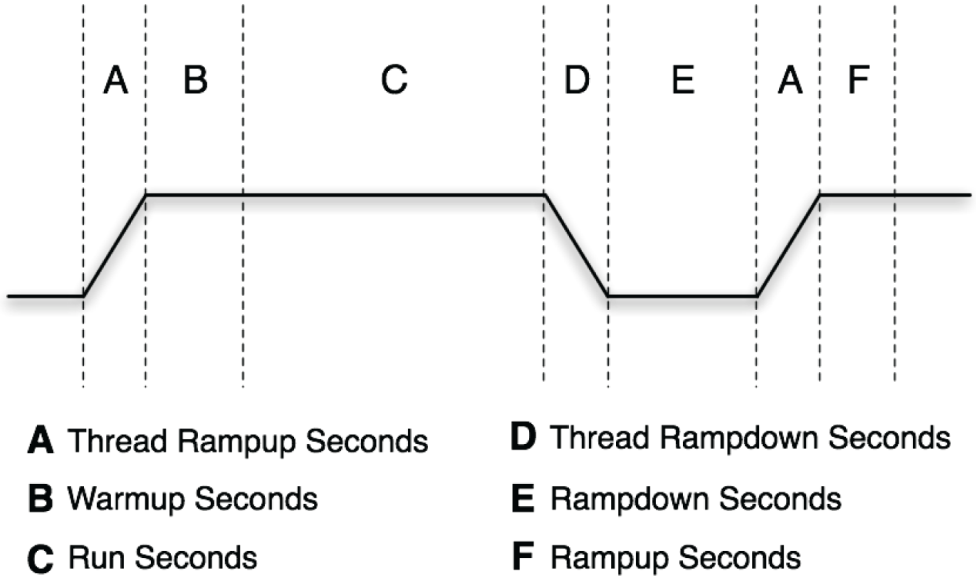

Figure 4.1: SPECweb 2005 test diagram.

- Phase D: the thread ramp-down period is simply the inverse of A. It is the period during which all load-generating threads are stopped. Although load generating threads are still making requests to the server during this interval, all recording of results will have stopped at the end of the run period.
- Phase E: the ramp-down period is the time given to the client and server to return to their "unloaded" state. This is primarily intended to assure sufficient time for TCP connection clean-up before the start of the next test iteration.
- Phase F: the ramp-up period replaces the warm-up period, B, for the second and third benchmark run iterations. It is presumed at this point that the server's cache is already primed, so it requires a shorter period of time between the thread ramp-up period and the run period for these subsequent iterations in order to reach a steady-state condition.

SPECweb Markov chain

A Markov Chain is a random process characterized as memoryless: the next state depends only on the current state and not on the sequence of events that preceded it. This specific kind of "memorylessness" is called the Markov property.

As stated before, SPECweb adopts this models of property for specifying the path of the user in the system emulated. As illustrated in Figure 4.2, the execution flows through the chain according to different values of probability, obtained from the analysis of users behavior in real systems. SPECweb implements the chain's edges with HTTP requests (GET, PUT, POST, ...) to the web server and each state is a single web page.

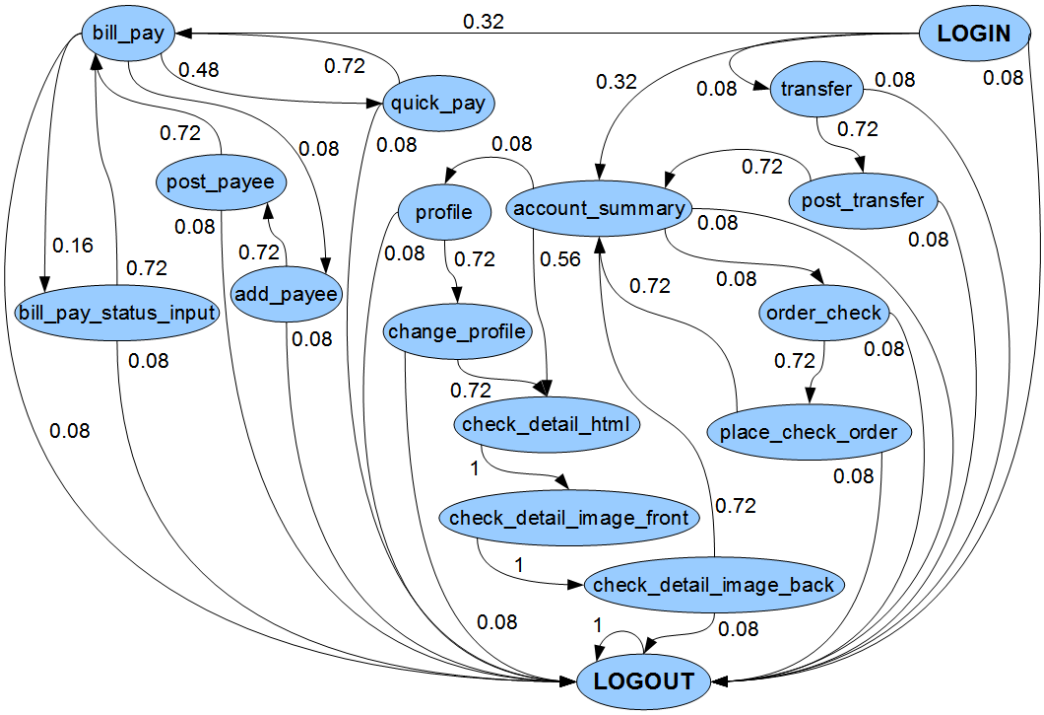

Figure 4.2: SPECweb banking test Markov chain.

4.5.2 JMeter extension

Initialization of the test environment

The first step we had to implement is an analysis of the startup phase of SPECweb. This phase includes the initialization of the web server, of the backend simulator and their interfacing. In order to do this we used the `.fcgi` files of SPECweb to initialize and prepare the Cloud instances to the new JMeter test environment.

Behaviour adaptation

One important feature of SPECweb tests is the preconfigured number of `SIMULTANEOUS_SESSIONS` in the configuration file, that it cannot be dynamically changed at execution time. One of the goal of our tool is to play on JMeter's features, in order to have a test with a varying number of simulated users. To do this we adopted the package of the JMeter Plugins project [53], that let us to defined a variable number of users for each single execution of a test.

Markov chain translation

In our tool we translate the SPECweb Markovian test design, describing the paths of a test execution and designing the correspondent configuration file with a thread group in JMeter. At run-time, a particular path will be chosen according to its probability in the Markov chain of its workload in SPECweb.

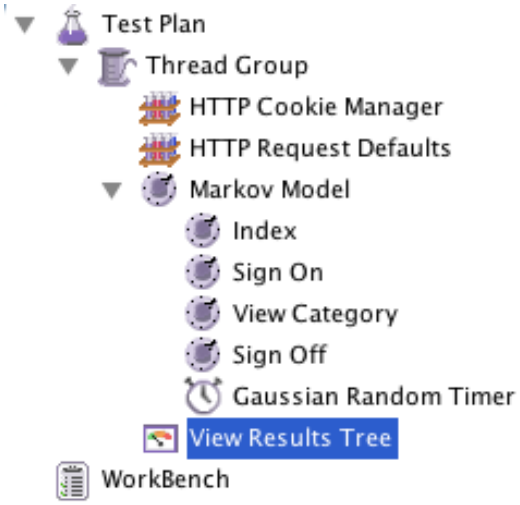

Figure 4.3: Markov chain example in JMeter.

The original JMeter provided by Apache does not include any feature to implement a Markov chain, so we adopted a plug-in developed by the University of Kiel, called Markov4JMeter [60], that allowed us to define the states of SPECweb's chain within the JMeter client (Figure 4.3) and the behaviours with simple `.csv` files containing the transitions probabilities as, for example, in Listing 4.1.

Listing 4.1: Sample file of behaviors

```
,‘‘Index’’,‘‘Sign On’’,‘‘View’’,‘‘Sign Off’’,‘‘$’’
‘‘Index*’’,0,0.2,0.8,0,0
‘‘Sign On’’,0.2,0,0.6,0.2,0
‘‘View’’,0,0.2,0.6,0.2,1
‘‘Sign Off’’,0,0,0,0,1
```

4.5.3 SPECmeter

Therefore, we developed a tool, called SPECmeter, in order to manage the functionalites reported above, to automatize and to make simpler the deployment of our injection tests on Amazon EC2 services.

For the sake of clarity in Figure 4.4 is represented the new Cloud architecture that will be automatically instantiated for each SPECmeter test.

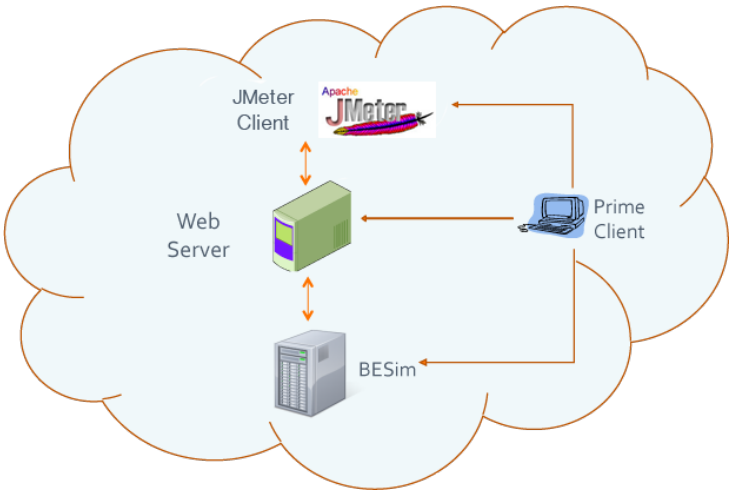

Figure 4.4: SPECmeter architecture.

Through the Amazon's API, the tool is able to start and configure the instances we need to run the benchmark. As in Figure 4.5, the Prime Client launches one or more instances, based on the number of simulated users, that hosts the JMeter Client, with proper configuration settings, and another two instances configured respectively as SPECweb web server and BeSim.

When everything is ready, SPECmeter launches the test and starts to monitoring and logging the virtual machines in terms of usage, traffic, number of requests, response time and other statistics useful to understand the performance of the Cloud environment.

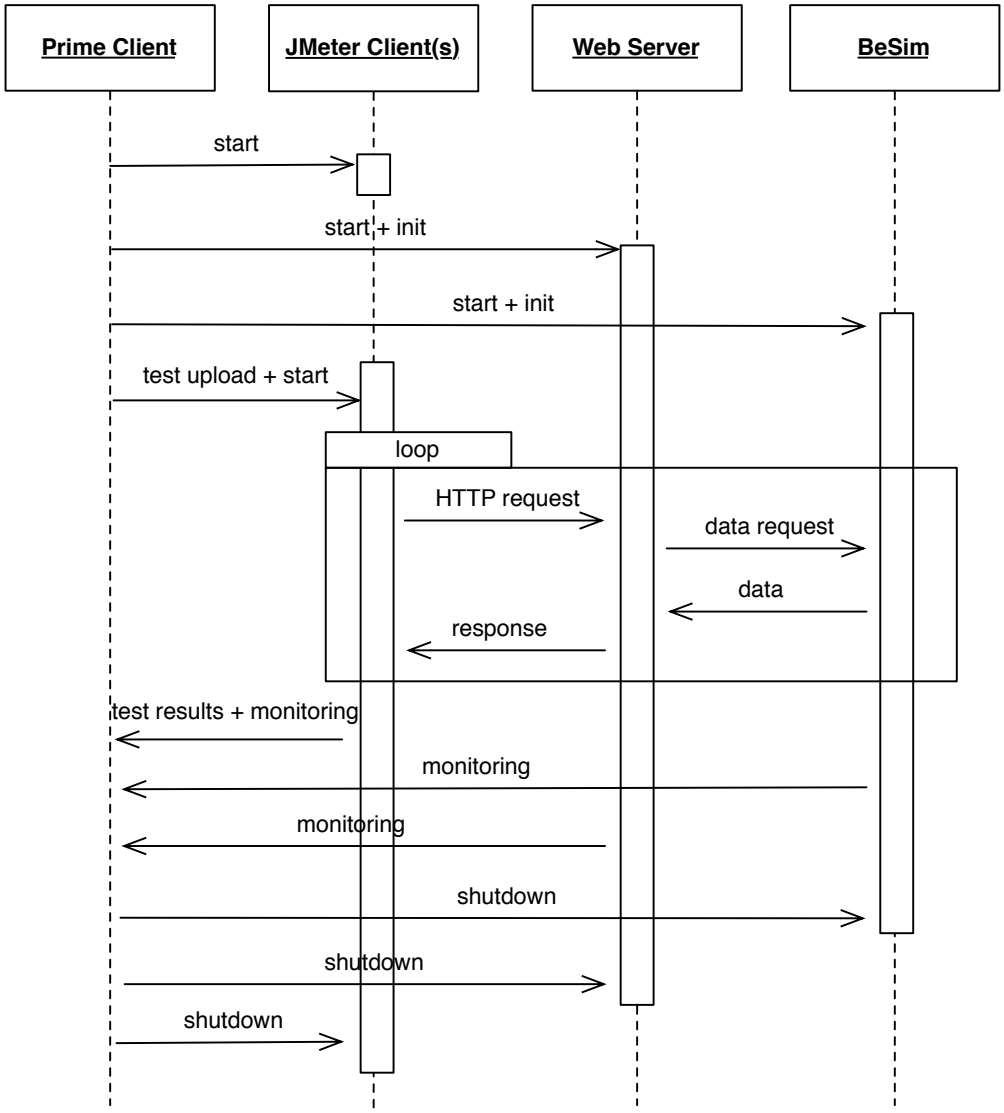

Figure 4.5: SPECmeter test automation sequence diagram.

At the test completion, the Prime Client downloads and parses results from Amazon instances and stores them in a proper folder. Then, if there are no further tests to run, it shutdowns all the instances used.

SPECmeter implementation

Our tool is a Python application that uses a set of libraries to interface the host with Amazon and the servers we previously deployed in the IaaS.

The first one is BOTO, a library able to use the API provided by Amazon. We used it start or shutdown the Amazon instances and to enable the Cloud-Watch monitoring service on them. To manage the SSH connections with our virtual machines we used the Paramiko (aka python-ssh) and the Crypto libraries are adopted for the private keys usage. With these we are able to configure the servers, upload and download files and launch the tests (Figure 4.5).

4.6 Cloud analysis tool

In this section we describe the tools we have developed in order to create the necessary data to perform analysis, run algorithms and then collecting results for studies that will be performed in Chapter 5.

AMPL works with three type of files: (i) the data files (`.dat`) that contains the whole database of parameters and input collections for the analysis; (ii) the mathematical models are contained in `.mod` files; (iii) functions, algorithms and connections among the analysis phases are contained in `.run` files, in a nutshell these files coordinate the run and solution of optimization problems.

Each analysis shares model files, but since we want to emulate different analysis, different environments and understand how the algorithms operate in a variety of situation, we develop a set of tools that can be called in order to automatically generate files that fits with AMPL standards in order to make analysis simple, fast and automatic.

The algorithm choice is not only concerning the procedure described in Chapter 3, but also about the alternative algorithms that will be proposed in the next chapter in order to compare them with our method.

4.6.1 Cloud analysis tool class diagram

The tool for the automation of the analysis is composed by four parts (see Figure 4.6). The main part is the so called *Simulation Manager* (SM), that is able to coordinate all the other components, manage the analysis phases, collect and analyze results, and finally store logs in the proper folder.

Afterwards, *Data Generator* (DG) is the set of scripts that generates the `.dat` files for the simulation. Changing the proper parameters these scripts create the necessary sets and the associations between these, and generate the traffic Λ_k for each application k, discussed in detail in Chapter 5.

The SM does not interact directly with CPLEX, but only with AMPL, managing the data loading, handling the model and selecting the algorithm

needed. AMPL, once set in the correct manner, can interrogate and cooperate with the solver independently, until the end of the analysis.

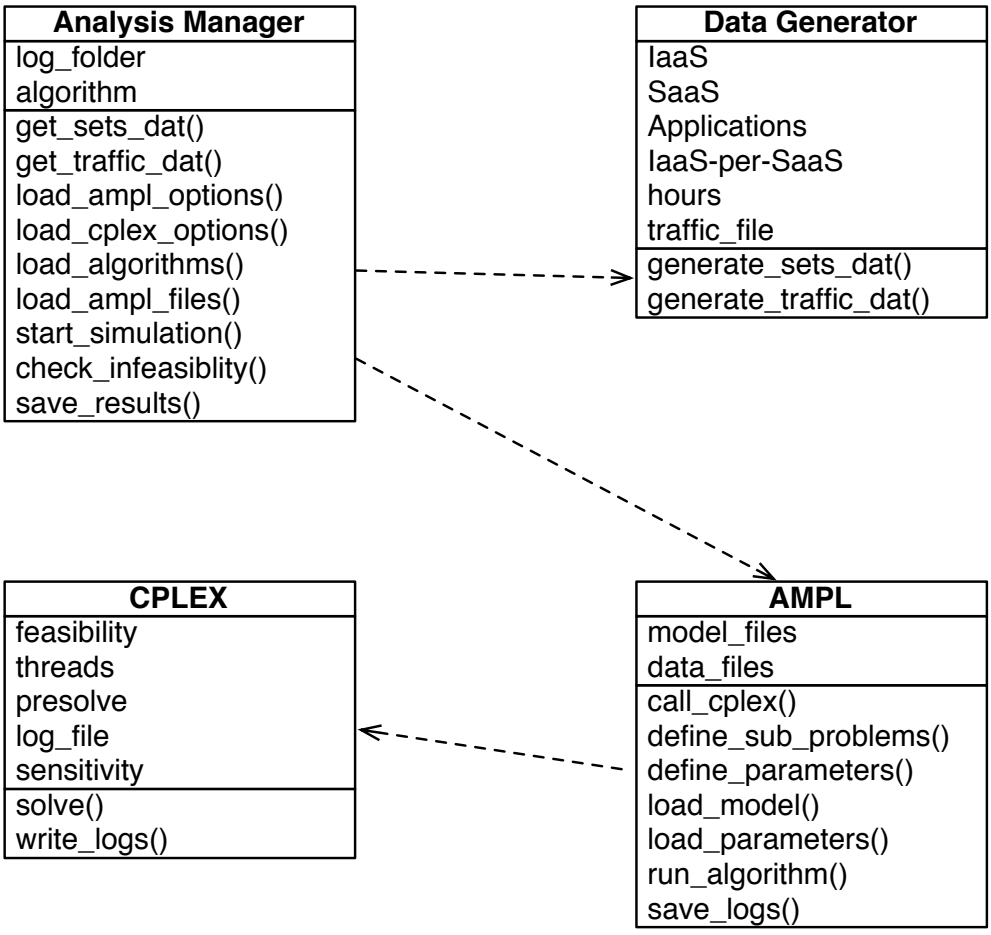

Figure 4.6: Cloud analysis tool class diagram.

4.6.2 Cloud analysis tool sequence diagram

After the description of the classes, we can now describe how they interact during the simulation (see Figure 4.7).

In the first phase the SM queries the DG with the desired cardinality of the instance simulated, the time horizon and the real data file to parse for the traffic generation. Once these files are ready, the SM initialize AMPL, loads CPLEX options, model and data files.

When the initialization phase is finished, the SM selects the algorithm, loads it into AMPL and starts it. Therefore the loop between AMPL and CPLEX solver is started until an equilibrium is found. Reached the final result, the SM analyzes it, reporting and logging an error message if it not feasible, which means that the equilibrium achieved does not meet all constraints. Finally, once everything is checked, results and logs of the simulated instance are saved in the proper folder.

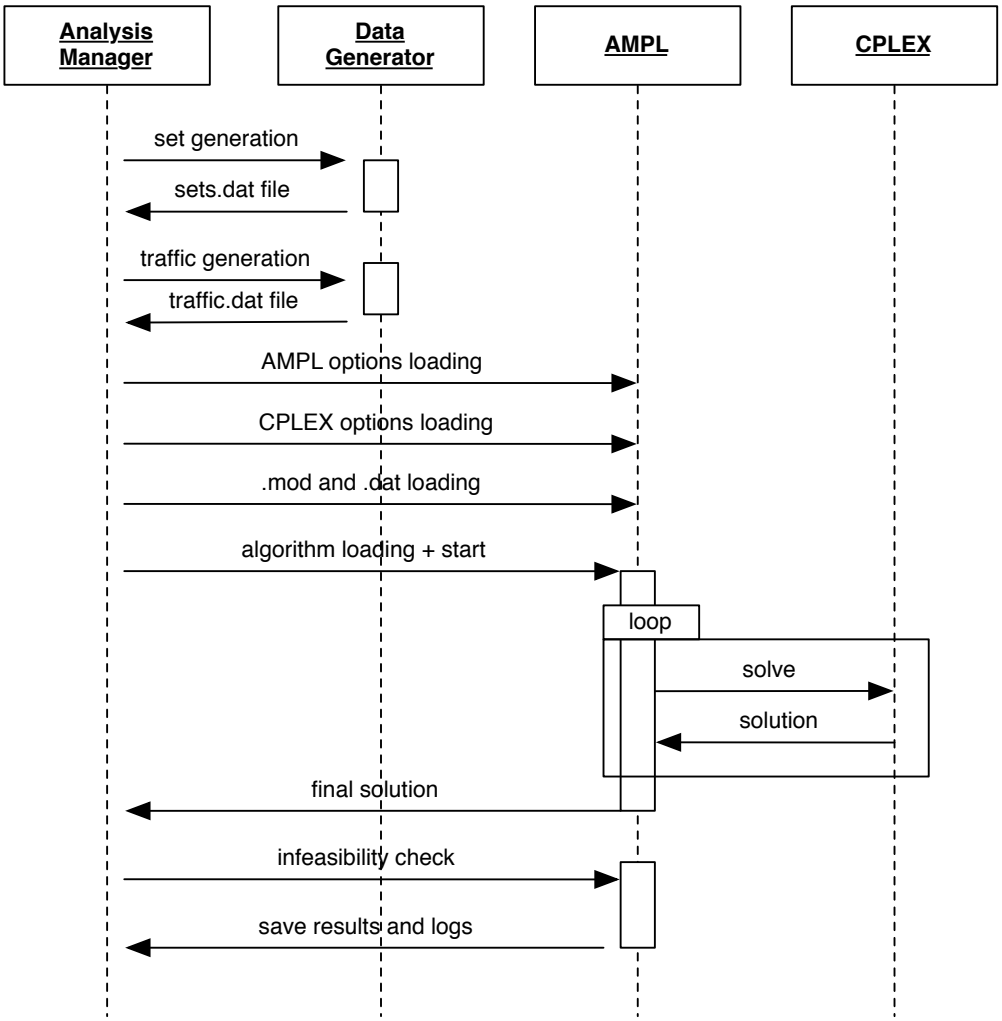

Figure 4.7: Cloud analysis tool sequence diagram.

Chapter 5

Experimental results

In this chapter we will present the experimental results achieved performing analyses based on the algorithm presented in Chapter 3.

Section 5.1 describes the experiments settings. In Section 5.2 the scalability of our approach is evaluated. Therefore Section 5.3 is dedicated to describe how we compare our solutions in terms of efficiency by considering alternative algorithms that can be used for service provisioning. Alternative algorithms are presented in Section 5.4, while their comparison is reported in Section 5.5. Finally in the last part, Section 5.6, we analyze the benefits that can be achieved by SaaS when hosting applications on multiple IaaSs.

5.1 Design of experiments

The analyses performed in this chapter are intended to be representative of a real Cloud environment. We therefore need to know the performance guaranteed by a typical Cloud provider, we need to simulate how the SaaS providers will rely upon the various IaaS and it is also important to replicate the trend of traffic during the day, both in terms of source and intensity.

Section 5.1.1 shows how we have assessed the range of the performance parameters and prices for the experiments. Mapping parameters between IaaSs and SaaSs are explained in Section 5.1.2, moreover in Section 5.1.3 details concerning time and worldwide distribution of traffic are reported.

5.1.1 Parameters generation

In order to evaluate the performance of a Cloud environment we used SPECmeter on a WS application deployed on an Amazon EC2 instance, as described in Chapter 4.

During our tests we keep the same architecture with one web server, deployed in a medium-size standard on demand instance in the US East region in North Virginia, but we varied the number of users with the purpose of getting a range of values for the service rate μ_{ki} and the queueing delay D_{ki}. Figure 5.1 and Figure 5.2 show a typical pattern of the parameters under analysis collected during a test. The maximum service rate and the queueing delay have been evaluated with the least squares method using response time and the Web server utilization collected during the test, as in [14]. The maximum percentage error on the average response time estimation is less than 20%.

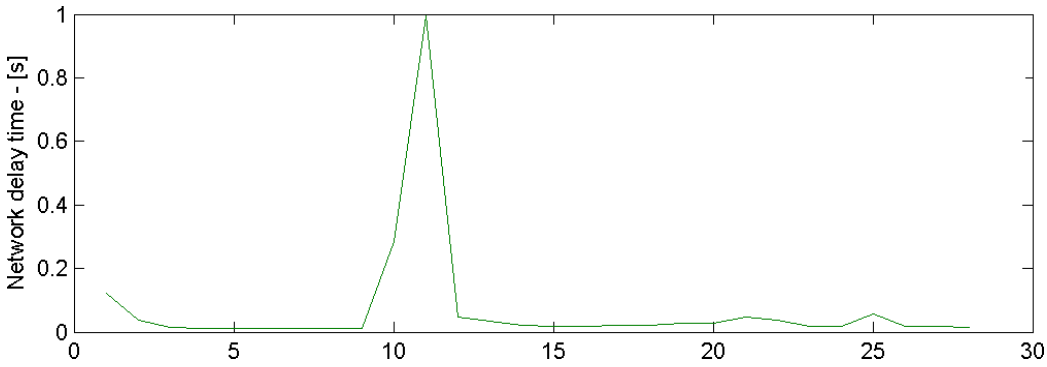

Figure 5.1: Queueing delay time.

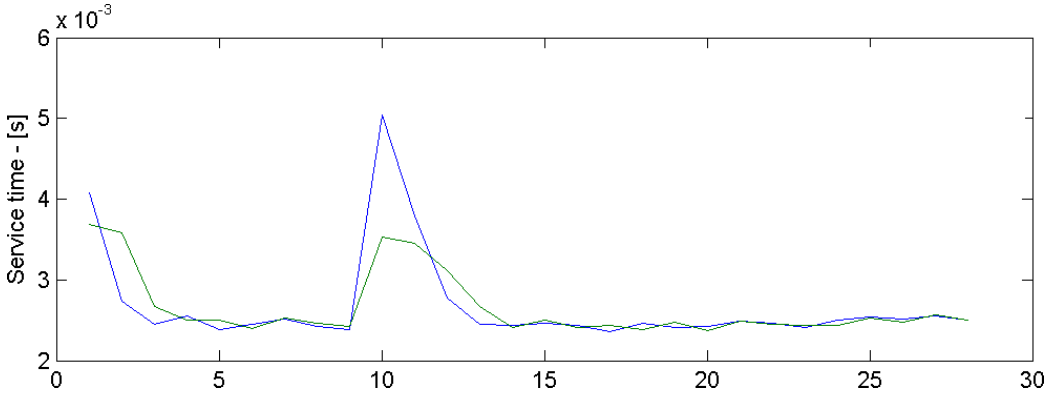

Figure 5.2: Service time.

Given these results we came to a range of $[200, 400]$ req/s for the service rate and a delay of $[0.001, 0.05]$ s for the queueing center.

Therefore, regarding the Cloud environment sizes, we have considered a large set of randomly instances obtained varying the model parameters according to the ranges reported in Table 5.1, as in other literature approaches

[18] [7] [56] and according to commercial fees applied by IaaS Cloud providers [9].

μ_{ki}	$[200, 400]$ req/s	σ_{ji}^{Max}	$[0.013, 0.047]$ \$/h
$\overline{R}_k$	$[0.025, 0.1]$ s	ρ_i	$[0.048, 0.076]$ \$/h
D_{ki}	$[0.001, 0.05]$ s	δ_i	$[0.12, 0.175]$ \$/h

Table 5.1: Performance parameters and time unit costs.

5.1.2 SaaS to IaaS mapping

A crucial part for the creation of analyses environment concerns the generation of $\mathcal{I}$, $\mathcal{S}$, $\mathcal{A}$ and corresponding sets $\mathcal{I}_j$, $\mathcal{S}_i$ and $\mathcal{A}_j$. Furthermore choosing the cardinalities of these sets the associations between IaaS, SaaS and WS application need to be defined accordingly. Therefore the *Data Generator*, as shown in Chapter 4, is started with desired cardinalities and parameters of associativity between IaaS and SaaS in order to create a suitable `.dat` file for the analysis.

Scalability and algorithm efficiency comparison analyses have been performed with different $\mathcal{S}$ and $\mathcal{A}$ sizes, as will be explained later, in the worst possible scenario where we consider $|\mathcal{I}| = 10$ and $|\mathcal{I}_j| = 3$ for all $j \in \mathcal{S}$, which means that each SaaS can host applications on 3 different IaaSs. Otherwise in multiple IaaS analysis is executed changing the $|\mathcal{I}_j|$ from 1 to 3 for all $j \in \mathcal{S}$, and keeping fixed the other instances cardinalities, with the aim of understanding how this association parameter affects the Cloud environment.

Therefore the DG generate through a "shuffle function", using the Knuth-Fisher-Yates algorithm, a set $\mathcal{S}_2$ of SaaS with a different order than $\mathcal{S}$. In this manner, during the execution of the algorithms we are sure that the resolution order of SaaS players does not affect the final equilibrium found.

5.1.3 Traffic generation

We simulated a Cloud environment distributed all over the world, using real data in terms of both the local and temporal distribution of requests.

With this feature we can emulate a real provider that hosts applications with different peak hours, different requests sources which means that the average capacity allocation can highly vary during the day depending on the hosted WS applications nature.

Time distribution of requests

Regarding the simulation of requests that arrive to the various applications, which means the Λ_k traffic distributed all over the IaaSs in the game, we used actual measurements coming from a large website that however, for privacy reasons, wants to remain anonymous.

The data are sampled every 10 minutes and overall our long trace lasts for 12 days. Therefore we parsed the source files, we adapted them to the hourly sampling time, and for each application k we applied a multiplier and a white noise to each sample, that can differentiate the workload of multiple classes, as in [14], [56].

Afterwards, in Figure 5.3 and 5.4, are represented respectively the daily and weekly distribution of requests of our source, while Figure 5.5 display a workload of a class with Greenwich Mean Time zone generated through the DG (see Chapter 4). A low workload occurs during the first hours of the day; the traffic then increases up to reach the peak during the working hours, before falling back in the night, cyclically, for each day of the week.

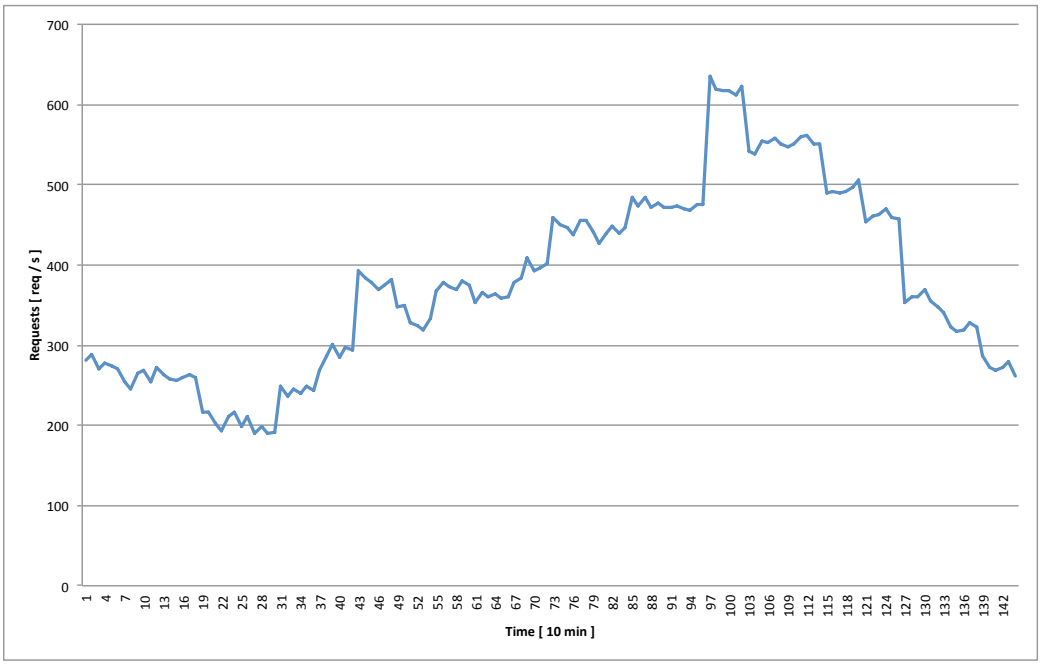

Figure 5.3: Daily time distribution of requests.

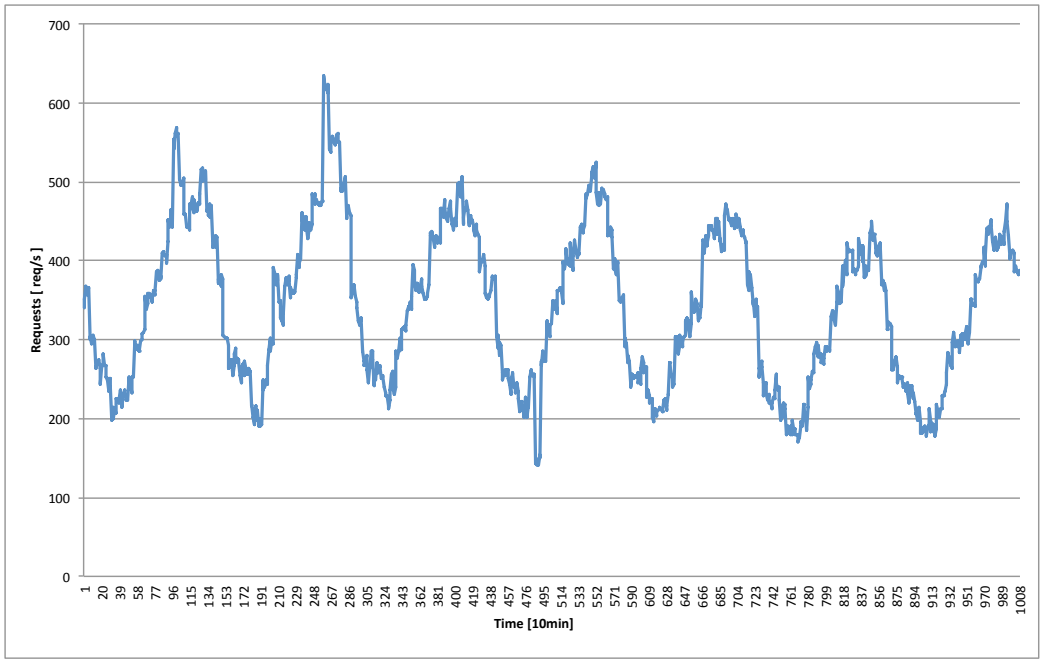

Figure 5.4: Weekly time distribution of requests.

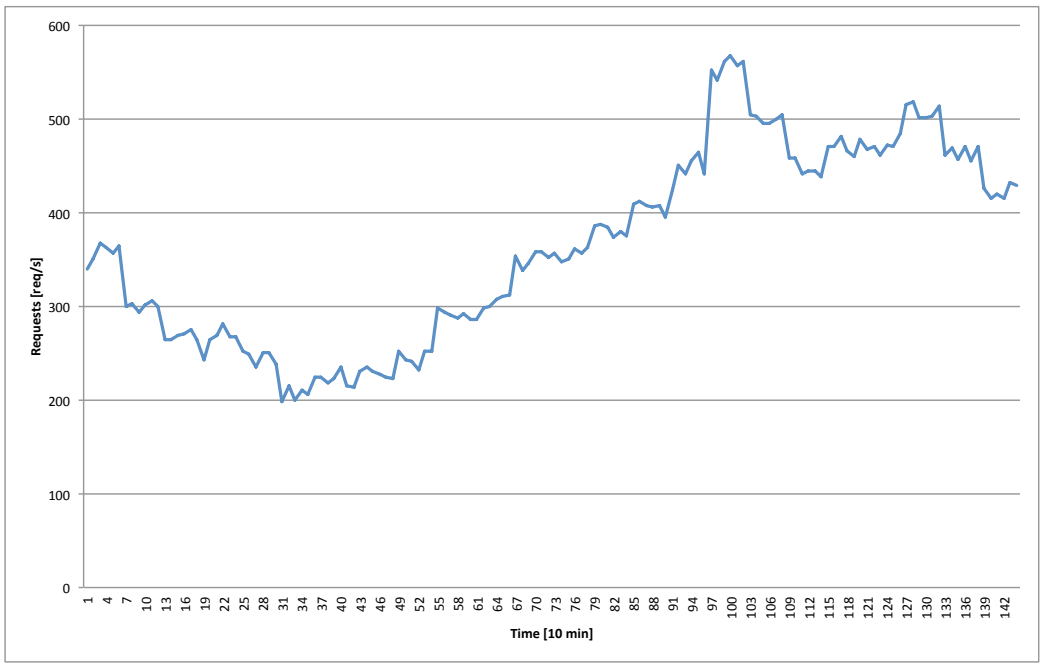

Figure 5.5: Generated daily time distribution of requests.

Worldwide distribution of requests

In order to create realistic scenarios we used statistics of a website reached from every part of the planet, i.e., Facebook, simulating the requests distribution that they published in their annual report [82].

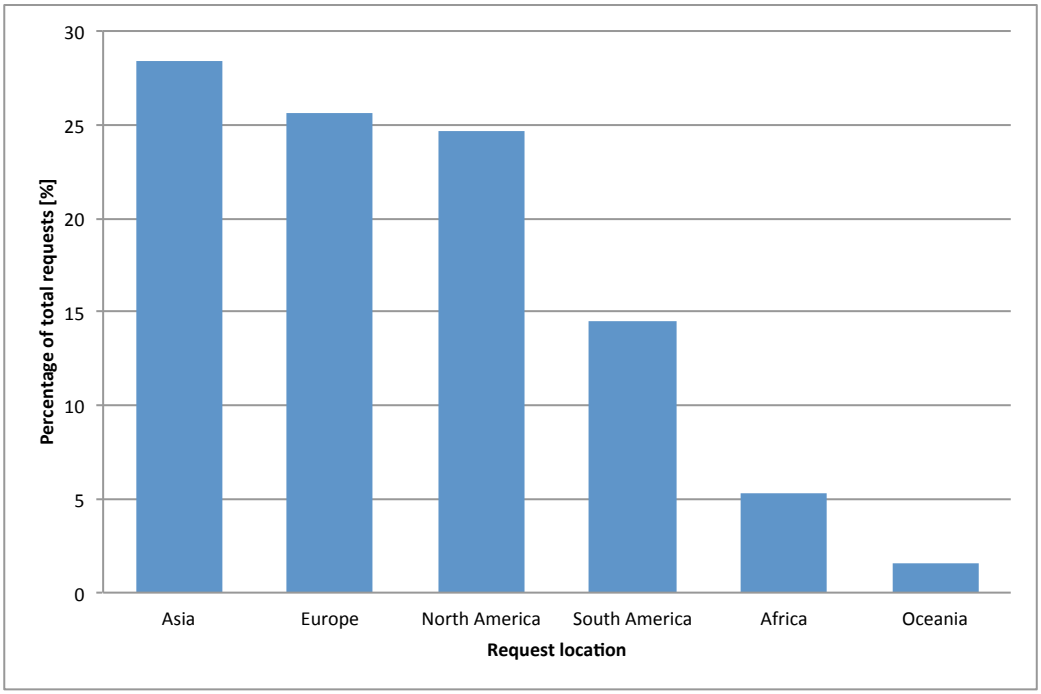

Figure 5.6: Worldwide distribution of requests.

Each application is then localized on a statistical basis, as shown in Figure 5.6. According to the assigned location, the daily pattern of arrival rate is adjusted to the correct time zone. In this way we can have the same site with WS applications with different peak hours, and even worst we can have multiple overlapping peak hours during the same day to manage.

5.2 Scalability analysis

To evaluate the scalabity of our resource allocation algorithm, presented in Chapter 3, we have considered a very lager set of randomly generated instances according to the parameters and procedures described above. The average values reported in this section have always been computed by considering 10 instances with the same size. The number of SaaS providers has

been varied between 100 and 1000, the number of applications (evenly shared among SaaSs) between 1000 and 10000.

The scalability of our approach, formulated in Section 3.4, has been evaluated performing tests on a Virtualbox virtual machine based on Ubuntu 12.04.1 LTS server (GNU/Linux 3.2.0-33-generic x86_64 kernel) running on an Intel Xeon Nehalem dual socket quad-core system with 32 GB of RAM.

Table 5.2 reports the average computational time required by each SaaS to run the algorithm presented in Chapter 3 for problem instances with different sizes of SaaS $\mathcal{S}$ and applications $\mathcal{A}$ sets and in the worst case regarding the cardinality of the IaaS set and the associativity sets, i.e., $|\mathcal{J}| = 10$ and $|\mathcal{J}_j| = 3$.

Instance size ($\|\mathcal{S}\|$, $\|\mathcal{A}\|$)	**Time (s)**
(100, 1000)	2.09
(200, 2000)	3.72
(300, 3000)	5.84
(400, 4000)	7.49
(500, 5000)	9.55
(600, 6000)	12.57
(700, 7000)	15.37
(800, 8000)	17.88
(900, 9000)	20.21
(1000, 10000)	22.32

Table 5.2: Distributed algorithm for identifying GNE execution times.

Given the linear interpolation shown in Figure 5.7, and the coefficient of determination $R^2 = 0.994$ calculated in relation to the collected data, we can state that the proposed method execution time per SaaS scale linearly with respect to the size of the instance.

Usually in Cloud system resource allocation is performed periodically on a hourly basis, [7], [19], [23]. Therefore we can affirm that the proposed method is very efficient and can be used at run-time since it can solve problem instances of maximum size in less than one minute.

5.3 Equilibria efficiency

The solution algorithm proposed has been evaluated for a variety of system and workload configurations as stated before. This section is devoted to present the metrics we used to understand the quality of our approach. Be-

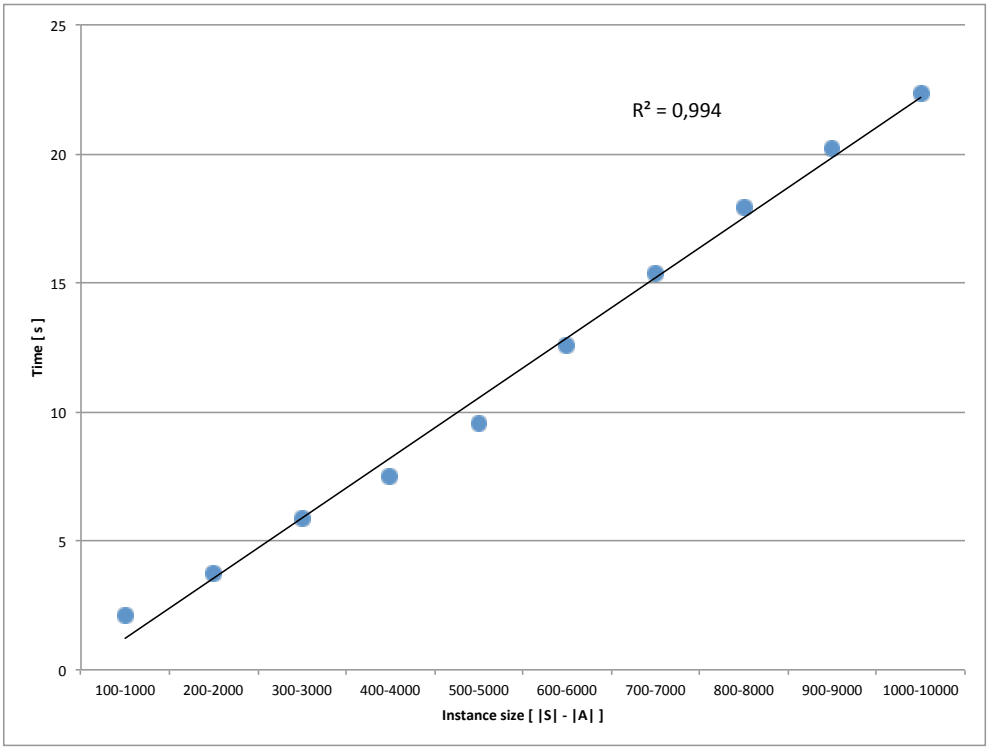

Figure 5.7: Distributed algorithm for identifying GNE scalability.

fore explaining which metrics we adopted, we explain how the social optimum is achieved and how we used it as reference measurement.

5.3.1 Social optimum problem

The social optimum problem (see Appendix A.4) is equivalent to a problem with the potential as objective function subject to the same constraints reformulated from (3.17) to (3.22) for the best-reply problem.

$$\min_{r_{ki},d_{ki},s_{ki}} \sum_{j\in\mathcal{S}} \sum_{k\in\mathcal{A}_j} \sum_{i\in\mathcal{I}_j} \rho_i\, r_{ki} + \delta_i\, d_{ki} + \sigma_{ji}^{Max}\, s_{ki} -$$

$$- T\,\nu_k \left[\mu_{ki} - \frac{1}{(\overline{R}_k - D_{ki})}\right] (r_{ki} + d_{ki} + s_{ki})$$

s.t.

$$\text{constraints (3.17) - (3.22).}$$

We denote with $(\widetilde{r}, \widetilde{d}, \widetilde{s})$ the social optimum solutions and with $(\overline{r}, \overline{d}, \overline{s})$ the equilibrium solutions found using the best-reply method.

5.3.2 Price of Anarchy and Individual Worst Case

The *Price of Anarchy* (PoA) is a concept in game theory that measures how the efficiency of a system degrades due to selfish behavior of its agents. As said, usually the efficiency is some function of the outcomes. In our case it is represented by the game potential Π. This indicator is defined as the ratio of the objective function value of a Nash equilibrium of a game and that of social optimal outcome. Note that this measure implicitly assumes that players successfully reach some GNE.

In order to evaluate the solutions of the algorithms the efficiency has been measured in terms of the PoA and of the *Indivual Worst Case* (IWC) evaluated as:

$$PoA = \frac{\Pi(\overline{r}, \overline{d}, \overline{s})}{\Pi(\widetilde{r}, \widetilde{d}, \widetilde{s})},$$
$$IWC_j = \max_{j \in \mathcal{S}} \frac{\Theta_j(\overline{r}, \overline{d}, \overline{s})}{\Theta_j(\widetilde{r}, \widetilde{d}, \widetilde{s})}.$$

Therefore, both PoA and IWC_j are a measure of the inefficiency due to IaaSs and SaaSs selfish behavior with the respect to the scenario where the social optimum is pursued. In particular, the IWC is a measure of the gap between the social optimum and the Generalized Nash equilibria achieved in the worst case by every single SaaS provider.

5.4 Alternative algorithms for resource allocation

In this section we will present a couple of algorithmic methods able to find a feasible solution for the service provisioning problem. The first one wants to mimic the behavior of a typical IaaS provider, so we take the perspective of Amazon, the global market leader. Furthermore, in the second sub-section we present another algorithm that can find a Generalized Nash Equilibrium exploiting constraint relaxation, but unlike the algorithm presented in Chapter 3 it benefits from resource rescaling.

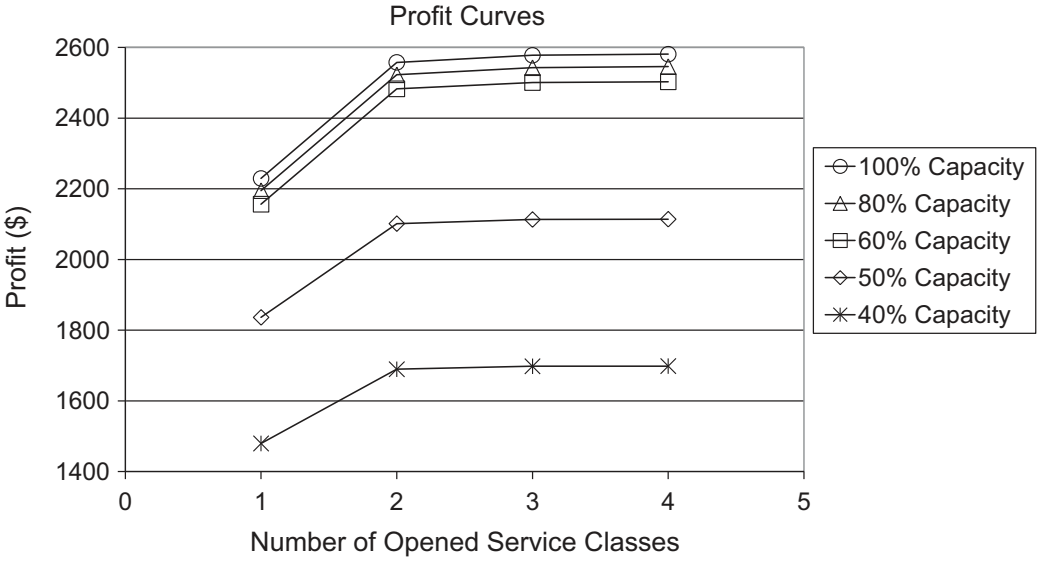

Figure 5.8: Impact of system capacity and service classes on the optimal profit, [59].

5.4.1 Heuristic

The heuristic presented in this section is intended to simulate the typical policies, based on CPU utilization thresholds, which can be implemented by SaaSs according to the required traffic and respecting the constraints determined in SLA contracts. As in other approaches considered in literature [98], [27], [108] we set the threshold at 60% CPU usage. Moreover results obtained in [59] (Figure 5.8 and Figure 5.9) show that by using 60% of the capacity the maximum ratio between profit and system usage is achieved.

On the contrary over the algorithm proposed in Chapter 3, that can be run in a distributed manner, the heuristic needs to be performed at IaaS site since it requires to know the whole set of decisions of the SaaS providers, in particular the σ_{ji}^{Max} prices for on spot VMs, as will be explained below.

This procedure is divided into several parts, according to the type of VMs available remaining in the IaaS infrastructure and its relative reliability. Each single step of the algorithm is composed of three nested cycles in order to satisfy every player of the game. The first concerns a loop for each individual SaaS $j \in \mathcal{S}$ in the allocation problem, the second concerns the IaaS $i \in \mathcal{I}_j$ on which every SaaS run its applications, and the innermost is performed once for each application $k \in \mathcal{A}_j$ belonging to the relative SaaS. Lastly, when each type of VM is assigned by the heuristic, the best reply function is performed for each SaaS player in order to achieve an equilibrium.

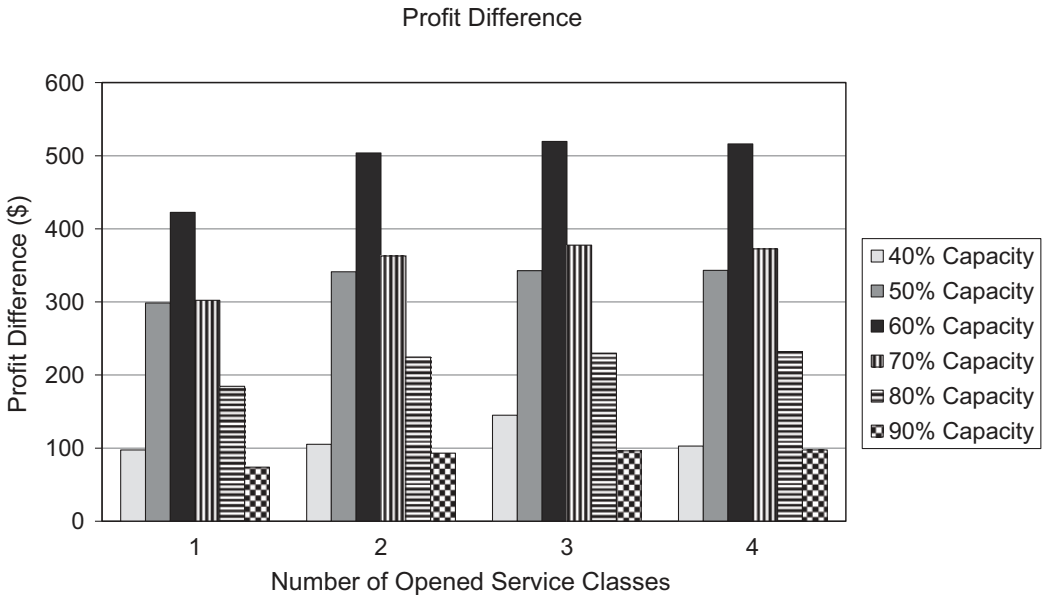

Figure 5.9: The difference of optimal profit with varied number of opened service classes, [59].

Initialization

The first step regards the initialization of the algorithm. The rationale carried out by the SaaS divides homogeneously the incoming traffic Λ_k according to the cardinality of $\mathcal{I}_j$, that's the number of IaaS on which the SaaS j hosts its applications. With the same reasoning, the number of reserved VMs r_{ki} is initialized according to the limit imposed by the R_{ji} upperbound.

Finally, we initialize a variable called $desired\_r_{ki}$ that represents the number of reserved VMs necessary at IaaS i for application k if we load at $U = 60\%$ the CPUs.

Algorithm 5.4.1: Initialization

for $j \in \mathcal{S}$ **do**
  for $i \in \mathcal{I}_j$ **do**
    for $k \in \mathcal{A}_j$ **do**
      $x_{ki} = \frac{\Lambda_k}{|\mathcal{I}_j|}$;
      $r_{ki} = \frac{R_{ji}}{|\mathcal{A}_j|}$;
      $desired\_r_{ki} = \frac{x_{ki}}{U \cdot \mu_{ki}}$;
    $sum\_desired\_r_i = \sum_{k \in \mathcal{A}_i} desidered\_r_{ki}$;

Reserved allocation

After the initialization obtained in the first step, the heuristic proceeds with a redistribution of reserved r_{ki} VMs proportionally to the number of real requests, respecting the relative upper bound R_{ji}.

According to the R_{ji} bound a new variable $reallocated\_r_{ki}$ is initialized as a fraction of the desired reserved VMs if the bound is violated, otherwise the $desidered_{ki}$ value is preserved.

Algorithm 5.4.2: Reserved allocation - part 1

```
8  for j ∈ 𝒮 do
9      for i ∈ ℐ_j do
10         for k ∈ 𝒜_j do
11             if (sum_desired_r_ki > R_ji) then
12                 reallocated_r_ki = (desired_r_ki · R_ji) / sum_desired_r_i ;
13             else
14                 reallocated_r_ki = desired_r_ki ;
```

The unassigned remaining VMs are stored in new sets of variables, called $residual_{ki}$, one set for each application k running at IaaS provider i, and reallocated as reserved VMs if still some applications is requiring them or later assigned if necessary in the next steps of the algorithm as different kind of VMs, both on spot or on demand.

Algorithm 5.4.3: Reserved allocation - part 2

```
15 for j ∈ 𝒮 do
16     for i ∈ ℐ_j do
17         for k ∈ 𝒜_j do
18             if (r_ki > reallocated_r_ki) then
19                 residual_ki = r_ki − reallocated_r_ki ;
20                 r_ki = reallocated_r_ki ;
21             else
22                 residual_ki = 0 ;
23             sum_residual_k = Σ_{i∈ℐ_j} residual_ki ;
24             if (r_ki < reallocated_r_ki) then
25                 if (residual_ki ≠ 0) then
26                     r_ki = r_ki + residual_ki / sum_residual_k ;
```

On spot allocation

After the reserved VMs assignment the algorithm switches to the on spot allocation, in order to try to meet throughput requirements while maintaining the IaaS service center bounds, due to the limited number of total VMs N_i available.

Algorithm 5.4.4: On spot allocation - part 1

```
for j ∈ 𝒮 do
    for i ∈ ℐ_j do
        for k ∈ 𝒜_j do
            desired_s_ki = max[0, (desired_r_ki − r_ki)] ;
            needs_N_i = Σ_{k∈𝒜_i} (desidered_s_ki + r_ki) ;
            sum_σ_j^Max = Σ_{i∈𝒮_i} σ_ji^Max ;
            available_s_ki = N_i − Σ_{k∈𝒜_j} r_ki ;
            sum_available_s_ki = Σ_{i∈ℐ_j} available_s_ki ;
```

The $desired_{ki}$ variable set is generated with the difference value between the desired and the assigned reserved VMs. Accordingly to this variables, $needs\_N_i$ is calculated as the sum of already allocated reserved instances plus the desired on spot. Following the reserved allocation, the $available\_s_{ki}$ VMs is calculated and used for on spot VMs rescaling.

Algorithm 5.4.5: On spot allocation - part 2

```
for j ∈ 𝒮 do
    for i ∈ ℐ_j do
        for k ∈ 𝒜_j do
            if (needs_N_i < N_i) then
                s_ki = desired_s_ki · σ_ji^Max / sum_σ_j^Max ;
            if (needs_N_i > N_i) then
                s_ki = s_ki · (available_s_ki / sum_available_s_ki) ;
```

Therefore, if the shared constraint (3.21) based on the N_i parameter is not active, a redistribution proportionally to the bidding price of the SaaS is performed. In this manner the IaaS reserve more resources for the customers willing to pay more. Otherwise a s_{ki} rescaling is performed with the aim to

satisfy the IaaSs physical limits.

On demand allocation

The last step of the allocations concerns the most expensive resource, the on demand VMs. If after reserved and on spot distribution the incoming traffic Λ_k is still unsatisfied, an amount of on demand VMs is turned on in order to satisfy at least the λ_k traffic requirements of the SLA contract.

The required number of VMs on demand is calculated as the difference between the amount of VMs required for satisfy the λ_k traffic, found through the performance parameters, and the on spot and the reserved already deployed.

Algorithm 5.4.6: On demand allocation

```
42 for j ∈ 𝒮 do
43     for i ∈ ℐ_j do
44         for k ∈ 𝒜_j do
45             if Λ_k.slack > 0 then
46                 d_ki = ⌈ ( λ_k / (μ_k − 1/(R̄_k − D_ki)) ) − (s_ki + r_ki) ⌉ ;
```

Heuristic evaluation

Algorithm 5.4.7: Best reply

```
1 for j ∈ 𝒮 do
2     for i ∈ ℐ_j do
3         for k ∈ 𝒜_j do
4             run Initialization (Algorithm 5.4.1);
5             run Reserved allocation (Algorithm 5.4.2, 5.4.3);
6             run On spot allocation (Algorithm 5.4.4, 5.4.5);
7             run On demand allocation (Algorithm 5.4.6);
8     solve SaaS_j problem (3.3.4);
```

Finally, after solving each VM type step, the heuristic solves the best reply solution starting from the obtained allocation. It finds a feasible generalized Nash equilibrium that satisfies all the constraints required, useful to evaluate the quality of the heuristic, through the metrics through metrics presented in Section 5.3.

5.4.2 Resource rescaling algorithm

The third algorithm we present is distributed and, as mentioned before, exploits the shared constraints relaxation as the other distributed method presented in Chapter 3. However in this algorithm, the reintroduction of the shared constraints in the solution of new sub-problems is handled in a different way as explained below.

Initialization

As for the algorithm presented in Chapter 3, being distributed, in the initialization step the SaaS providers send to the IaaS their bid $\sigma_{ji}^{Max} = \rho_i - \epsilon$.

Algorithm 5.4.8: Initialization

```
for i ∈ ℐ do
    for j ∈ 𝒮_i do
        σ_ji^Max = ρ_i − ϵ ;
```

Solution of relaxed problems

Each SaaS provider solves independently its individual minimization sub-problem without the shared constraints $h_i(y)$ with the other SaaS providers in the game:

$$\sum_{k \in \mathcal{A}_i} (r_{ki} + d_{ki} + s_{ki}) \leq N_i \qquad \forall\ i \in \mathcal{I}_j.$$

Algorithm 5.4.9: Solution of relaxed problems

```
for j ∈ 𝒮 do
    solve SaaS_j problem without shared constraints h_i(y) ;
for i ∈ ℐ do
    M_i = Σ_{j∈𝒮_i, k∈𝒜_j} (r_ki + d_ki + s_ki) ;
```

We then save M_i, that is the sum of the required VMs for every SaaS at IaaS i without the shared constraint in the model, useful for the next steps.

Stopping criterion

This part is the core of the algorithm and checks if the number of assigned VM by each IaaS $i \in \mathcal{I}$ is higher than the available at each service center

and, if it occurs, a rescaling of the number of VMs is performed in order to satisfy data centers bounds. The reiteration loop is based on the $\mathcal{I}_C$, which initially contains all the elements of the $\mathcal{I}$ set.

Algorithm 5.4.10: Stopping criterion

```
8  set I_C = I;
9  while I_C = ∅ do
10     for j ∈ S do
11         for i ∈ I_j do
12             if M_i ≤ N_i then
13                 remove i from I_C;
14             else
15                 for k ∈ A_j do
16                     r_ki = r_ki · N_i/M_i ;
17                     d_ki = d_ki · N_i/M_i ;
18                     s_ki = s_ki · N_i/M_i ;
19                 solve SaaS_j;
20     M_i = Σ_{j∈S_i, k∈A_j} (r_ki + d_ki + s_ki) ;
```

If the condition in line 12 is satisfied, the algorithm remove the i element from the $\mathcal{I}_C$ set. Otherwise every IaaS rescales the number of any kind of VMs of a ratio $\frac{N_i}{M_i}$ for each application $k \in \mathcal{A}_j$ and call CPLEX in order to solve the SaaS problem with the shared constraint. Therefore a recalculation of the M_i parameter is done and the algorithm restarts its loop until the $\mathcal{I}_C$ set is empty, which is equivalent to the fact that there are no IaaSs with more resources than what they can offer.

5.5 Algorithms efficiency comparison

Thanks to the criteria of evaluation presented in 5.3 we compare the algorithms in terms of the efficiency of the reached equilibrium.

Performance parameters and time unit costs have not been changed with respect to Table 5.1, but we focused on understanding what change in terms of efficiency when the IaaS i offers a different ϕ_i upper bound percentage of reserved instances compared with its total N_i resources, where

$$\phi_i = \frac{\sum_{j \in \mathcal{S}_i} R_{ji}}{N_i} .$$

Given that the data regarding IaaSs trade policies are not public, three values of ϕ_i have been considered in order to emulate a real Cloud environment: 10%, 30% and 50%.

With the purpose of comparing the algorithms in situations in which the problems of individual SaaSs are not separable, which means in those situations where the shared constraints are active, and consequently the PoA and the IWC are greater than 1, we carried out several tests with $|\mathcal{J}| = 10, |\mathcal{S}| = 100, |\mathcal{J}| = 1000$, and $|\mathcal{J}_j| = 3$ for each SaaS.

For practical reasons hereafter we rename procedure presented in Section 5.4.1 as *Heuristic*, the resource rescaling algorithm detailed in Section 5.4.2 as *Algorithm 1*, while as *Algorithm 2* the method described in Chapter 3.

The procedure we adopted to increase the difficulty of resource distribution corresponds to decrease the N_i size of each IaaS, step by step, until the allocation becomes infeasible. As shown in Figure 5.10, which depicts a representative example of the percentage of VMs usage during a peak hour, with an high value of N_i VMs available at each IaaS are assigned in a low percentage and consequently the shared constraints are not active. The problem becomes easy to solve since there is no competition among players and each single SaaS j problem can separated from the others and optimally solved. Note that in these cases PoA will be equal to 1.

Otherwise, diminishing the N_i value, the service provisioning problem becomes more complex, more stringent, shared constraints become active and SaaS players start to compete for resources assignment. As verifiable from *Re-optimizations* tables listed below, this situation induces a number of re-optimizations greater than zero. Re-optimizations are performed when, in both distributed algorithms, a first solution found with constraint relaxation assign a total number of VMs greater than what IaaSs can offer. Re-optimization represents the number of violations obtained while the *Stopping criterion* loops are performed (Algorithm 5.4.10 for Algorithm 1, Figure 3.3 for Algorithm 2). Therefore a rescaling in Algorithm 1 and bound changes in the Algorithm 2 are perfomed, and re-optimizations started till a feasible solution. On average Algorithm 2 computes more iterations, but gives, especially in hard cases close to infeasibility, better results both for PoA both for the IWC in any ϕ_i analyzed.

With respect to the heuristic we can see from PoA and IWC tables that in cases where SaaS problems are separable better results are achieved. In not challenging environments the reached equilibrium is equivalent to the social optimum, that correspond to better solution of 0.3% to 0.8% for the PoA and from 2% to 5% for the IWC, depending on the ϕ_i used, with regards to the Algorithm 1 and Algorithm 2.

The advantage of the heuristic compared to distributed algorithms is due

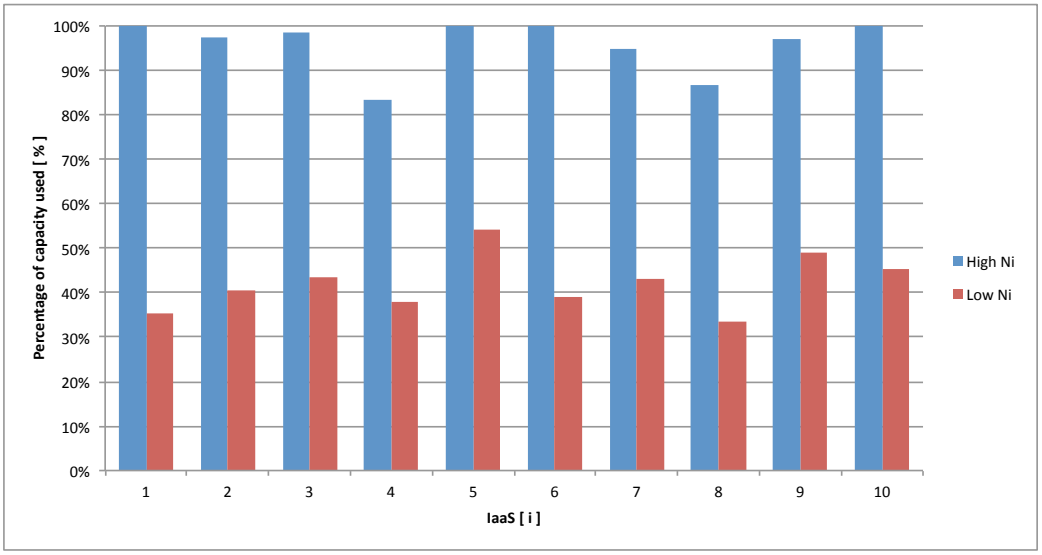

Figure 5.10: IaaS capacity usage on peak hours.

by the fact that is performed at the IaaS sites. This means that each SaaS knows the σ_{ji}^{Max} prices and unlike the distributed algorithms where, being executed at SaaS sites, the information concerning time unit cost offered by IaaS i for on spot VMs instances to other SaaSs must be approximated.

However, the heuristic lose its advantage when the problem becomes tighter and shared constraints start to be active and they make the solution process more complex. When the distributed algorithms start the re-optimization due to competition among SaaSs for the resources, the heuristic reaches a feasible solution but with unacceptable efficiency. The metrics show that the reached equilibria, depending on the ϕ_i, are 363% worst than the social optima problem with regard to the PoA and average worsening of 477% for the IWC of each SaaS.

Moreover in extreme environments with very low available N_i VMs per IaaS (e.g., Low N_i case in Figure 5.10) independently from ϕ_i used, where distributed algorithms require an high amount of re-optimizations but reach a feasible and very efficient solutions, the heuristic comes to infeasible results, baring out its limitations and showing benefits of using optimization models.

PoA			
N_i	**Heuristic**	**Algorithm 1**	**Algorithm 2**
95	1.00000	1.00343	1,00343
90	1.00051	1.00358	1.00358
85	1.00189	1.00366	1.00366
80	1.05331	1.00376	1.00376
75	1.36599	1.00382	1.00382
70	—	1.00430	1.00383
65	—	1.00461	1.00384
60	—	1.00484	1.00386

Table 5.3: Algorithms PoA comparison with $\phi_i = 0.1$.

IWC (Average over SaaS providers)			
N_i	**Heuristic**	**Algorithm 1**	**Algorithm 2**
95	1.00000	1.02614	1.02614
90	1.00081	1.02680	1.02680
85	1.03397	1.02717	1.02717
80	2.01429	1.02745	1.02745
75	4.77981	1.02756	1.02756
70	—	1.02862	1.02763
65	—	1.03149	1.02786
60	—	1.03271	1.02826

Table 5.4: Algorithms IWC comparison with $\phi_i = 0.1$.

Re-optimizations		
N_i	**Algorithm 1**	**Algorithm 2**
95	0.0	0.0
90	0.0	0.3
85	0.8	1.3
80	2.0	2.5
75	4.3	6.5
70	5.5	8.3
65	6.8	29.0
60	14.2	49.3

Table 5.5: Algorithms re-optimization comparison with $\phi_i = 0.1$.

PoA			
N_i	**Heuristic**	**Algorithm 1**	**Algorithm 2**
75	1.00000	1.00450	1.00450
70	1.00101	1.00405	1.00405
65	1.13108	1.00370	1.00370
60	—	1.00371	1.00357
55	—	1.00385	1.00307

Table 5.6: Algorithms PoA comparison with $\phi_i = 0.3$.

IWC (Average over SaaS providers)			
N_i	**Heuristic**	**Algorithm 1**	**Algorithm 2**
75	1.00000	1.03325	1.03325
70	1.02014	1.03129	1.03129
65	3.05352	1.02930	1.02930
60	—	1.03255	1.02815
55	—	1.02982	1.02715

Table 5.7: Algorithms IWC comparison with $\phi_i = 0.3$.

Re-optimizations		
N_i	**Algorithm 1**	**Algorithm 2**
75	0.0	0.0
70	1.0	1.3
65	1.0	1.8
60	3.3	4.5
55	8.8	22.3

Table 5.8: Algorithms re-optimization comparison with $\phi_i = 0.3$.

PoA			
N_i	**Heuristic**	**Algorithm 1**	**Algorithm 2**
70	1.00000	1.00888	1.00888
65	1.00066	1.00808	1.00808
60	2.05349	1.00721	1.00721
55	3.62520	1.00626	1.00626
50	—	1.00642	1.00568
45	—	1.00648	1.00396

Table 5.9: Algorithms PoA comparison with $\phi_i = 0.5$.

IWC (Average over SaaS providers)			
N_i	**Heuristic**	**Algorithm 1**	**Algorithm 2**
70	1.00000	1.04920	1.04920
65	1.01189	1.04661	1.04661
60	2.44790	1.04422	1.04422
55	4.33550	1.04090	1.04090
50	—	1.03801	1.03699
45	—	1.04689	1.03324

Table 5.10: Algorithms IWC comparison with $\phi_i = 0.5$.

Re-optimizations		
N_i	**Algorithm 1**	**Algorithm 2**
70	0.0	0.0
65	0.0	0.2
60	1.0	1.0
55	1.0	1.7
50	4.3	7.5
45	15.8	86.5

Table 5.11: Algorithms re-optimization comparison with $\phi_i = 0.5$.

5.6 Multiple IaaS analysis

This section is devoted to the study of the advantages, disadvantages and possible trade-offs that arise when a SaaS relies on more than one IaaS to host its applications using the Algorithm 2, which equilibria are the best in terms of efficiency as stated in the previous section.

Renting resources across multiple IaaS in our model means to change the cardinality of $\mathcal{I}_j$ sets. This parameter is handled by the *Data Generator* that, once properly configured, automatically create consistent `.dat` files for the desired environment prototype.

Afterwards we performed analyses with SaaS providers able to buy resources from one up to three IaaS at the same time and we compared them in terms of equilibria efficiency and payoff function value, that means the total price the SaaS has to pay to the IaaSs that host its applications.

In order to perform consistent analyses, the R_{ji} maximum number of reserved VMs that can be executed for the SaaS j at IaaS i has been properly configured in such a way that it can turn on an equal number of this type of VMs independently from how many IaaSs it can use for hosting its applications:

$$\sum_{i \in \mathcal{I}_j,\, |\mathcal{I}_j|=1} R_{ji} = \sum_{i \in \mathcal{I}_j,\, |\mathcal{I}_j|=2} R_{ji} = \sum_{i \in \mathcal{I}_j,\, |\mathcal{I}_j|=3} R_{ji}\,, \qquad \forall j \in \mathcal{S}\,.$$

As in Section 5.5 we used the parameters of Table 5.1 and we focused on the same three values for ϕ_i. In addition to results achieved on equilibria, we also examined the average value of the payoff function (PF), which symbolize in \$ the fee paid from the SaaSs to their IaaSs.

Before displaying and analyzing the results obtained in the multi-IaaS analysis, we take the point of view of a representative SaaS, with low or very high traffic periods during the day, in order to better understand what means to be able to rent resources from different providers.

If we take the perspective of a SaaS provider that has to satisfy the sum of the WS applications traffic supported, exemplified in Figure 5.11, we can found it in different situations.

In the first one, if its own IaaSs allow it, all the traffic can be satisfied only renting reserved and proportionally to the η_j parameter on spot instances (see Figure 5.12). In this way it can easily guarantee that all requests will be served and the availability will be granted to its customers.

Otherwise, if the SaaS meets restrictions in resource allocations due to the limited number of resources of the IaaS, or by other SaaS players asking for

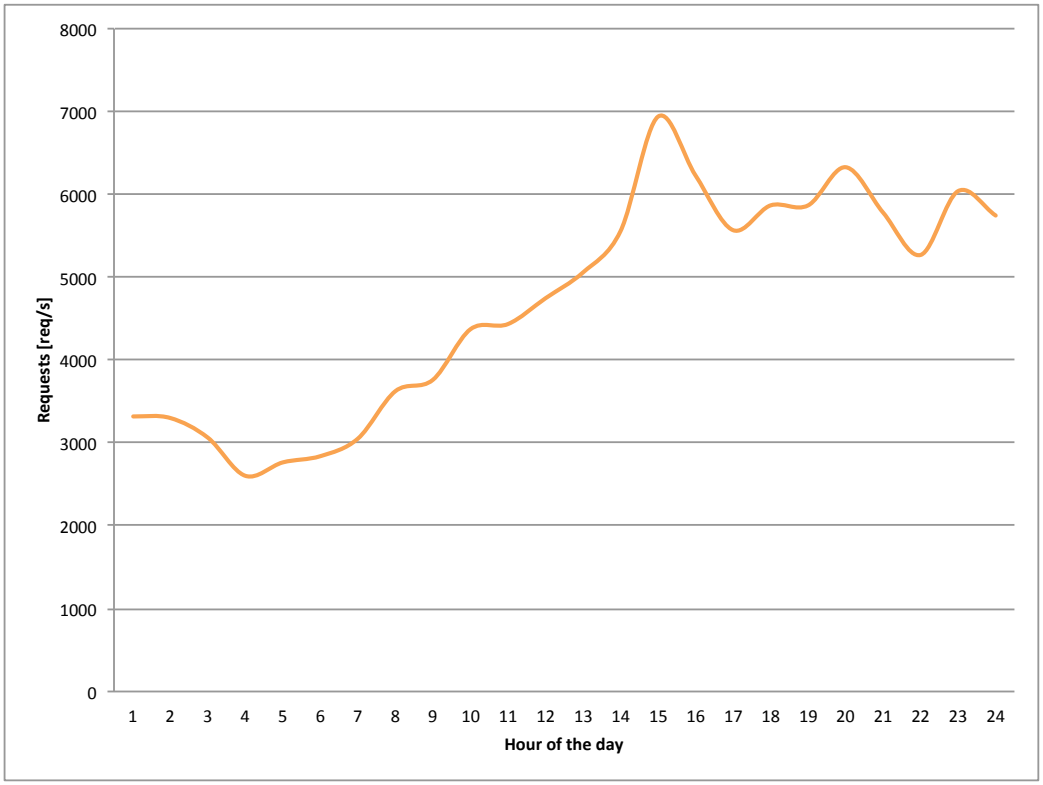

Figure 5.11: Traffic example of a single SaaS provider.

the same resources, or to the low R_{ji} amount of reserved instances assigned to its own VMs due to for example a low ϕ_i, it should acts a countermeasure.

In the case it profits by using more than one IaaS, it can starts VMs into another IaaS provider until the traffic is satisfied. Differently, if it not lies on multiple IaaSs or if its dedicated reserved are finished and on spot ratio is reached, it must turn on the most expensive VMs, the on demand ones, as shown in Figure 5.13.

Moreover, thanks to the fact that the SaaS player can benefit of multiple IaaSs, it can compete for the cheapest resources on multiple sites, in order to lower its costs. It can bid for lowest on spot resources, or start, depending on the availability, reliable VMs in the lowest price zone.

Another important aspect relates to the reliability that a SaaS can ensure. If it relies on a single IaaS, in case of downtime of the latter, it can not satisfy the incoming traffic. Otherwise if it uses a $\mathcal{J}_j$ set with more than one provider, it can move the traffic to its other suppliers avoiding interruption of services, which corresponds in a greater QoS for the SaaS's customers.

Clarified the advantages of the SaaS of being able to use *software layers*

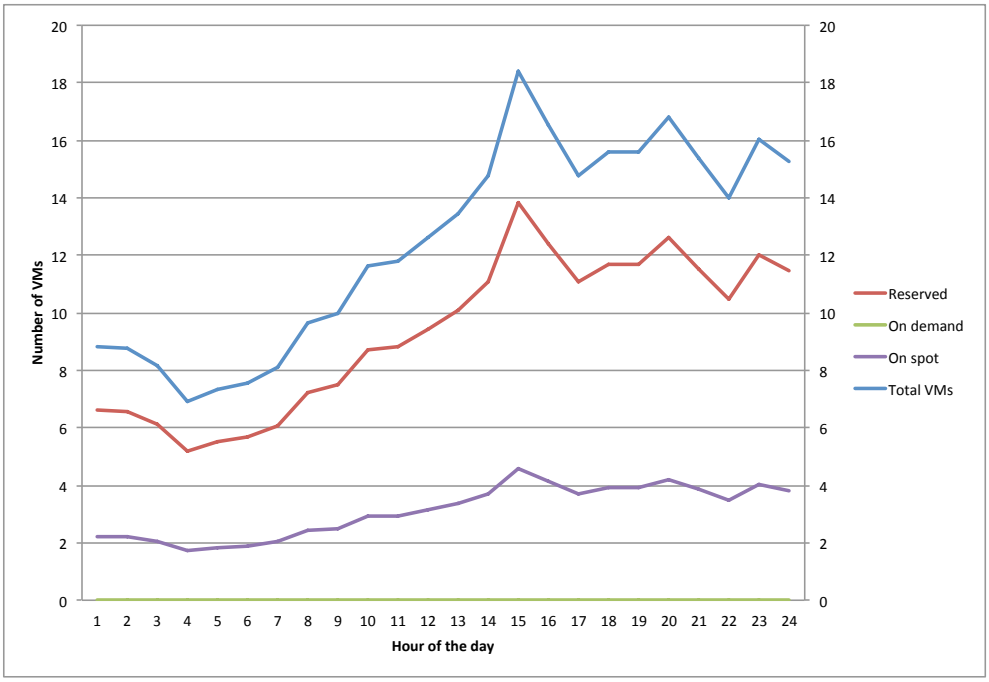

Figure 5.12: SaaS allocation with unlimited resources.

to deploy WS applications on more than one IaaS, we can analyze results where, with same incoming requests, we simulate SaaS relying to multiple IaaSs and we compare it with the single IaaS case. As noted in Section 5.5, in order to test the algorithm in different situations, we set the tests with the purpose of having a situation near to infeasibility, then with a limited number of N_i VMs at each IaaS in the peak hours. This corresponds to an average usage of 60% of total available VMS in non-peak hours, and close to 100% in every IaaS during high traffic periods.

The first category of analyses regards Cloud environment characterized by $\phi_i = 0.1$, which means that each single IaaS offers a maximum of 10% of its virtual machines as reserved. A summary of average results is reported in Table 5.12, where the value of the PF is expressed in $.

Immediately appears clearly the advantage of having more IaaS to exploit from the SaaS payoff function point of view. There is a saving of 39% with the support of another IaaS compared to the single-IaaS deployment, and up to 50% when the SaaS relies to three IaaS providers.

This great savings is due to the fact that the IaaS offering fewer reserved VMs, in the single-IaaS case the SaaS is forced to turn on expensive on demand VMs that quickly increase the fees, which in most cases is better

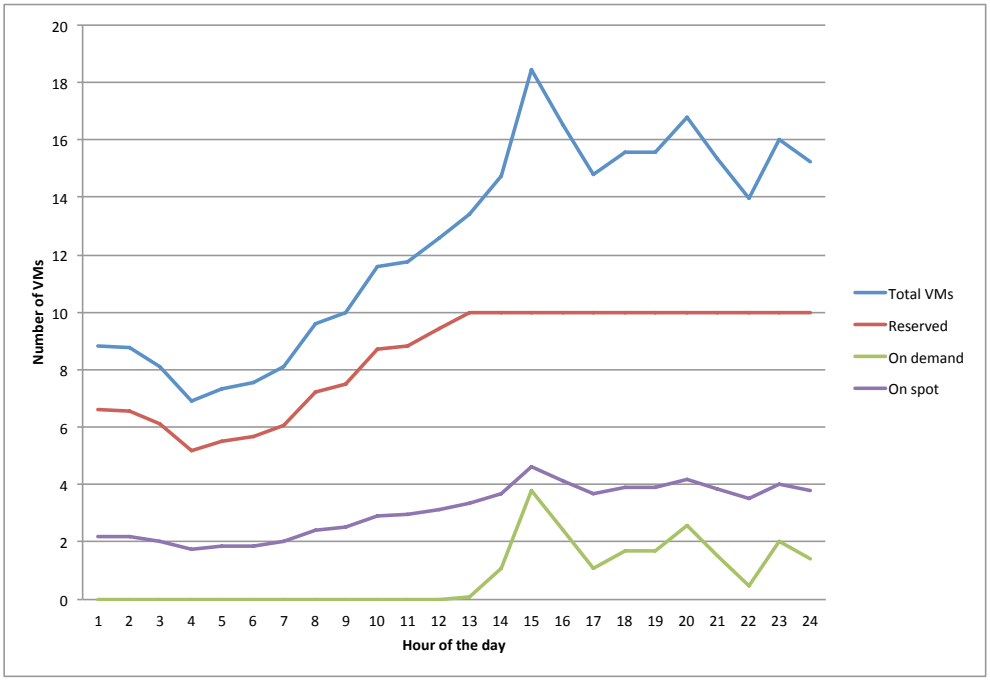

Figure 5.13: SaaS allocation with limited resources.

	$\lvert\mathcal{I}_j\rvert$		
	1	**2**	**3**
PoA	1.0000	1.0074	1.0161
IWC	1.0000	1.0412	1.0593
PF	45.5513	27.6885	22.7805

Table 5.12: Multi-IaaS analysis results with $\phi_i = 0.1$.

than paying the penalties for not satisfied SLAs contracts.

However, as graphically show in Figure 5.14, there is a trade-off between the average PF value and its average efficiency in the worst case reported from the IWC, that is 5.93%. The same loss of quality is reported from the PoA, that with $\phi_i = 0.1$, is equal to a maximum loss of 1.61% compared to the social optimum with $|\mathcal{I}_j| = 3$.

For the sake of clarity in Figure 5.15 a set of hundred simulated payoff functions is reported with the single-IaaS PF value normalized to 100% and compared to Multi-IaaS results. Results shows that the gap with $\phi_i = 0.1$ is very significant in every SaaS provider, particularly between the single and the multiple IaaS approach.

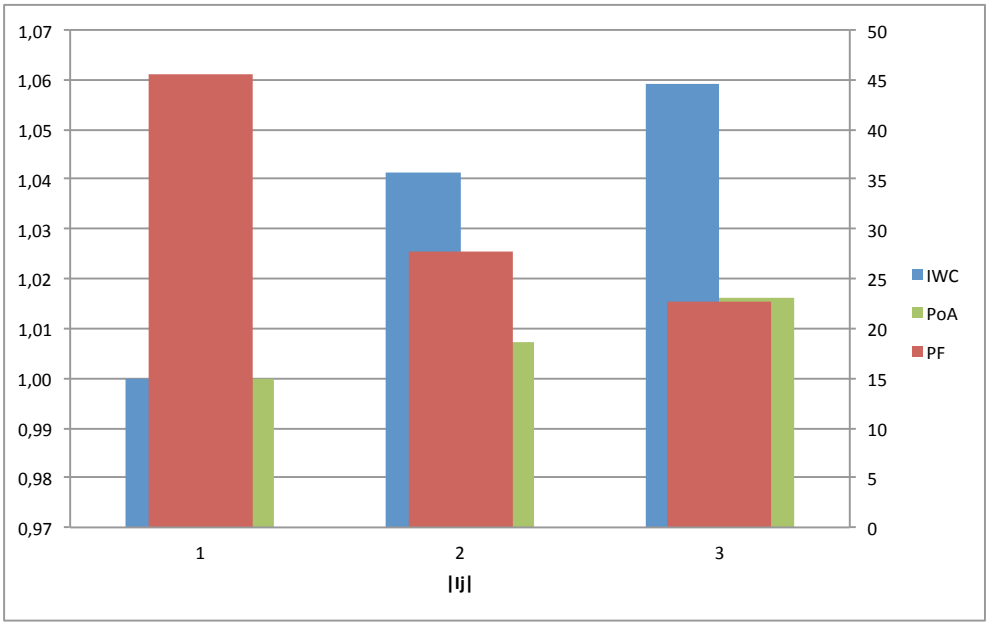

Figure 5.14: Multi-IaaS analysis results with $\phi_i = 0.1$.

Therefore we performed a series of tests with $\phi_i = 0.3$, collecting results as summarized in Table 5.13.

	$\|\mathcal{I}_j\|$		
	1	**2**	**3**
PoA	1.0000	1.0078	1.0127
IWC	1.0000	1.0521	1.0629
PF	33.0845	25.4269	22.8917

Table 5.13: Multi-IaaS analysis results with $\phi_i = 0.3$.

As verifiable from Figure 5.16 and Figure 5.17 there is still an average 23% savings deploying application on two and 30% on three IaaSs. The gap between the single and multiple IaaS decreases thanks to an higher amount of reserved VMs on each IaaS provider, and consequently a lower quantity of on demand instances is required. This fact is reflected in a large lowering of the PF in the case of single IaaS, a lower decrease in the case of $|\mathcal{I}_j| = 2$, and a slight increase in the average price to be paid in the case of three IaaS, due to increased competition for the resources present in every single provider.

Finally, we tested the Cloud environments with $\phi_i = 0.5$, that means half of the VMs available at each IaaS providers may be of reserved type.

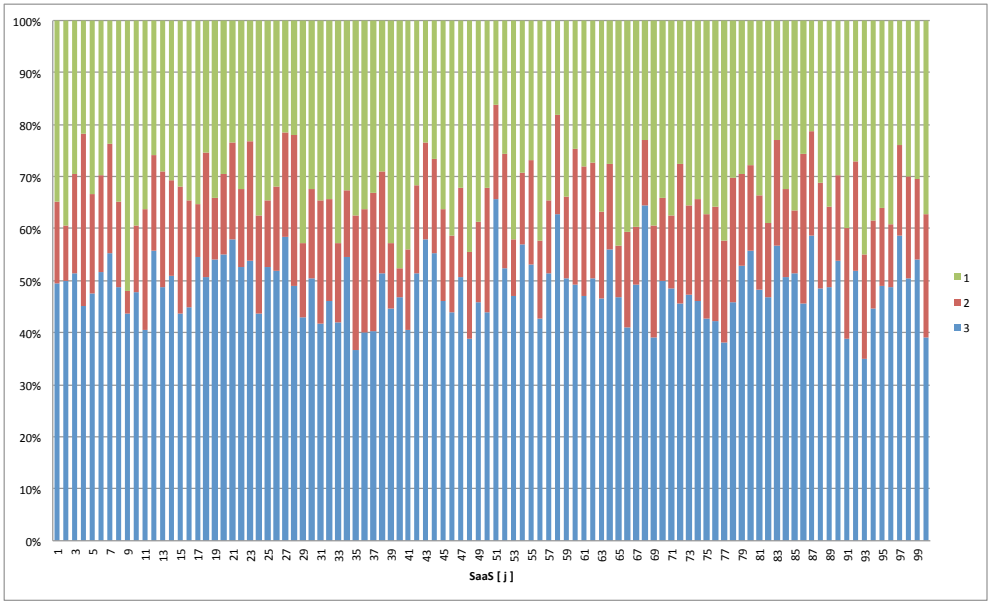

Figure 5.15: Multi-IaaS payoff function comparison with $\phi_i = 0.1$.

As the reader may imagine at this point, the gap between single and multiple Cloud deployment is even decreased. However, the results (Table 5.14) show that there is still significant advantages, with an average savings starting from 16% up to 22%.

	$\lvert\mathfrak{I}_j\rvert$		
	1	**2**	**3**
PoA	1.0000	1.0081	1.0119
IWC	1.0000	1.0534	1.0598
PF	29.5973	24.8274	22.9073

Table 5.14: Multi-IaaS analysis results with $\phi_i = 0.5$.

Moreover the equilibria efficiency is improved, as summarized in Figure 5.18. There is a drop to 1.19% for the PoA and 5.98% for the IWC. This shows the effectiveness of our approach in situations where competition for the VMs is strong and the allocation becomes even more complex given the increase of SaaS players competing for the same resources.

With such high value of ϕ_i occur cases (see Figure 5.19) where three providers deployment is more expensive than hosting applications on two IaaSs. This happens with customers willing to buy on spot VMs that, however, in the case in which they have already been sold and being available in

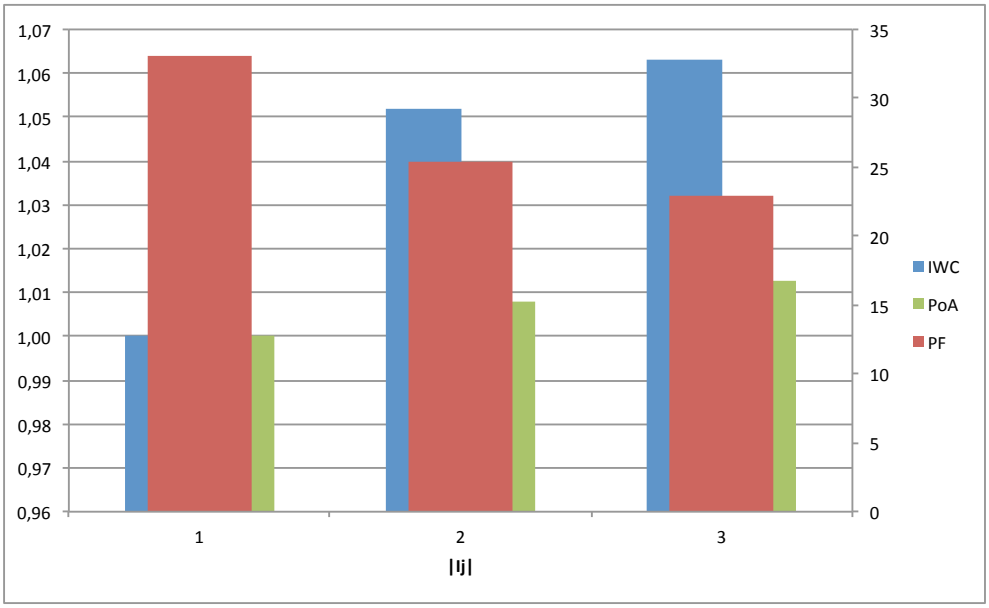

Figure 5.16: Multi-IaaS analysis results with $\phi_i = 0.3$.

smaller quantities due to the increased number of reserved, SaaS providers are obliged to start on demand VMs, raising consequently their fees.

Given the results obtained, it is clear that exploiting more than one IaaS providers simultaneously is beneficial for the SaaS. In terms of profits SaaSs will have savings ranging from a minimum of 16% up to 50%. Regarding the equilibrium achieved in the worst case the inefficiency is equal to less than 2% for the social problem and up 6% for the single player.

Finally, it is important to point out the considerable advantages concerning the reliability that the SaaS provider can guarantee to its customers, which increases with the number of used IaaSs.

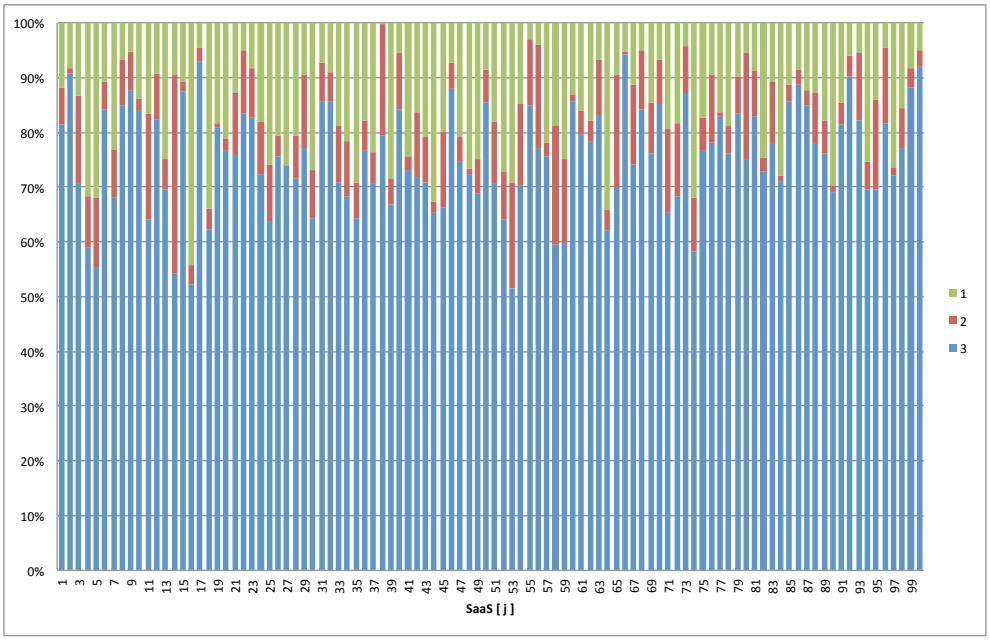

Figure 5.17: Multi-IaaS payoff function comparison with $\phi_i = 0.3$.

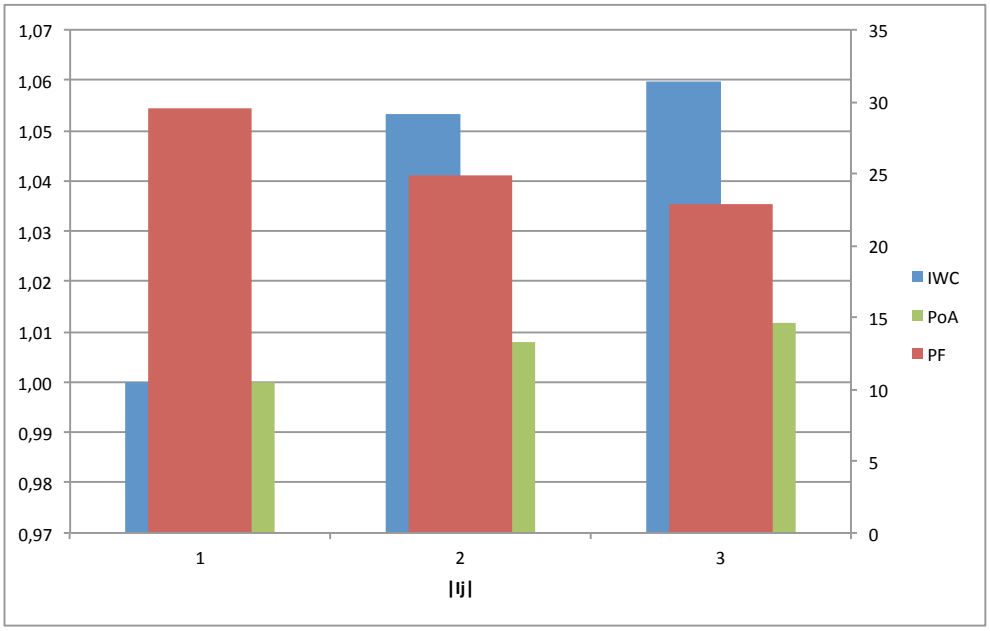

Figure 5.18: Multi-IaaS analysis results with $\phi_i = 0.5$.

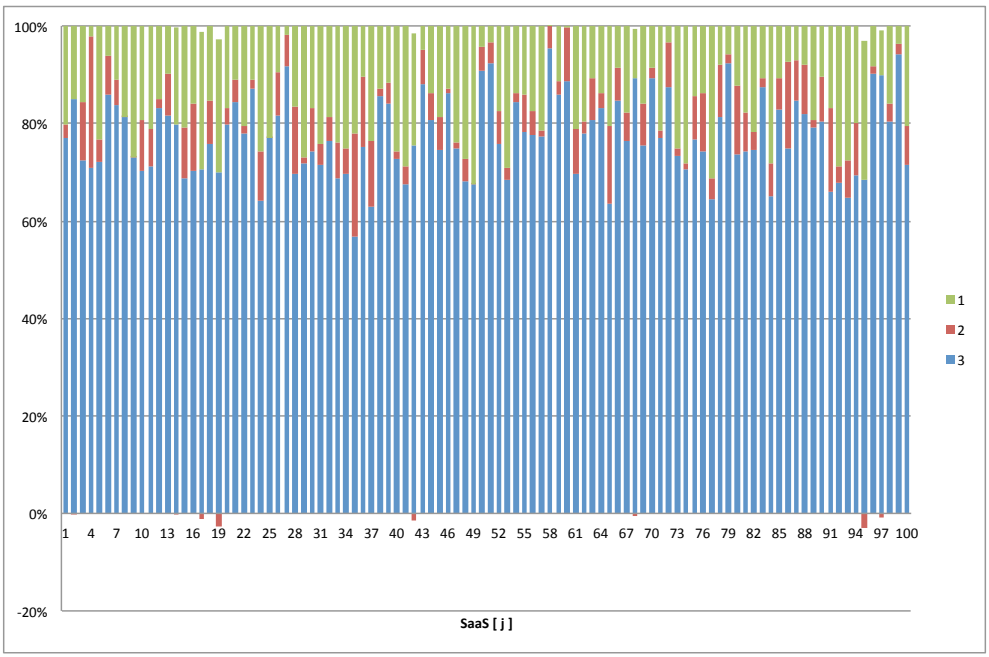

Figure 5.19: Multi-IaaS payoff function comparison with $\phi_i = 0.5$.

Chapter 6

Conclusions

As Cloud-based services become more numerous and dynamic, resource provisioning becomes more and more challenging, especially when decisions can be affected by the action of others, not only by own actions, hence requiring game-theory approaches. Indeed, in any time instant resources have to be allocated to handle effectively workload fluctuations, while providing quality of service guarantees to the end users.

The overall goal we addressed in our thesis is the minimization of the costs associated with the allocated virtual machine instances in multiple IaaSs, while guaranteeing QoS constraints. To achieve this purpose we have proposed a game-theoretic approach for the run-time management of IaaSs provider capacities among multiple competing SaaSs. The cost model consists of objective functions which include revenues and penalties incurred depending on the achieved performance level and infrastructural costs associated with IaaSs resources. Therefore a distributed algorithm for identifying Generalized Nash Equilibria have been presented and its termination in a finite number of iterations has been demonstrated.

Thanks the AMPL language and the CPLEX solver, the effectiveness of our approach have been assessed by performing a wide set of analyses under multiple workload conditions. Realistic workloads created from a large website statistics and performance parameters estimated on an industrial benchmarking deployed in the Cloud have been used.

A number of different scenario of interest have been considered. Systems up to thousands of applications can be managed very efficiently in a fully distributed manner. Our algorithm found efficient GNE, for a hourly basis resource allocation, in less than a minute, proving to be perfectly suitable for run-time provisioning.

A comparison with utilization based state-of-the-art techniques and a rescaling algorithm shows that our solution outperform alternative methods

proving better results in terms of equilibrium efficiency both as regards the PoA both the IWC of each SaaS. In addition, our algorithm, achieve an efficient GNE under heavy workload conditions while thresholds based heuristic finds infeasible results due to the inability to manage SaaSs competition.

Finally analyses showed clear SaaS benefits while exploiting multiple IaaS deployment of applications and redistribution of traffic. SaaSs can have an average savings up to 50% compared to single IaaS architectures. The equilibria achieved are close to Cloud optimum, with inefficiency less than 2% for the social problem and of 6% for individual player in the worst cases.

Future work will extend the proposed solutions to consider multiple timescales for performing resource allocation from few minutes to one hour. Additionally short-term solutions will be based on receding horizon techniques.

Appendices

Appendix A

Game theory and generalized Nash equilibrium problem

A.1 Game theory in the Cloud Computing

Game theory has found its applications in numerous fields such as Economics, Social Science, Political Science, Evolutionary Biology. Over the last years this branch of applied mathematics has found its applications also in Computer Science. Because of the success of the Internet and the revolution in Information Technology, the nature of computing has changed making it possible to commoditize the components such as network, computing, storage and software. In the new paradigm, there are multiple entities (hardware, software agents, protocols etc.) that work on behalf of different autonomous bodies (such as a user, a business etc.) and provide services to other similar entities. Many geographically distributed autonomous entities interact with each other through the Internet and provide various services working for their respective owners to achieve their individual goals (maximize their individual payoffs), as opposed to obtaining a system optima (that is socially desirable). Therefore, it is important to study some computer science problems under a game-theoretic model.

As for Cloud Computing, its social, economic and strategic structure make it impossible to apply classical optimization approach in order to model its mathematical problems. Indeed, some of them, like the service provisioning problem, are perfectly suitable to Noncooperative Game Theory tools.

In this section we present the basic concepts of Game Theory and provide its application in different domains. In particular, we define what is an equilibrium and explain the different notions of it. Finally, we illustrate the theoretic results necessary to study the problem proposed in this work.

A.2 Definition of Game

Non-cooperative Game Theory is the study of problems of conflict and cooperation among multiple independent decision-makers, which means the study of the ways in which *strategic interactions* among *economic agents* produce *outcomes* with respect to the preferences (or *utilities*) of those agents, where the outcomes in question might have been intended by none of the agents. Each actor pursues his/her own interests working independently and without assuming anything about what other players are doing. Moreover, he/she has to follow certain rules while making these moves and each one of them is supposed to behave rationally.
In the language of Game Theory rationality implies that every player is motivated by maximizing his own payoff irrespective to what other players are doing.

More formally:

Definition 1. *A non-cooperative game* Γ *in strategic form is a tuple* $\left\{N, \{X_i\}_{i\in N}, \{\Theta_i\}_{i\in N}\right\}$ *that consists of:*

- *a finite set of players* $N \equiv \{1, 2, ..., n\}$*, where* $n \in \mathbb{N}$*;*
- *a set of strategies* X_i *for every player* $i \in N$*, which is also called feasible set for player* i*;*
- *cost functions,* $\Theta_i : X_1 \times X_2 \times \cdots \times X_n \to \mathbb{R}$ *for each player* $i \in N$*.*

Moreover, we indicate with X*:*

$$X \equiv X_1 \times X_2 \times \cdots \times X_n \subseteq \mathbb{R}^M \tag{A.1}$$

the common strategy set, called feasible set or strategy space of game Γ*; every point* $\boldsymbol{x} \in X$ *represents the feasible strategies of the game. Note that* X *is supposed to be nonempty, closed and convex.*

The interpretation of the above definition is the following: every player i chooses simultaneously his variable $x_i \in X_i$ producing a feasible solution $\mathbf{x} = (x_1, x_2, ..., x_n)$. Then, every player has to pay a cost that is $\Theta_i(\mathbf{x})$.

With this context in mind, the challenge of a game is to understand when one choice is better than another for a particular player. The best strategy for a player in a game may be *pure* or *mixed*; the first when players

deterministically choose their moves, the latter when they randomly choose one out of many different strategies. Besides the types of interactions between players, a game can be classified according to the nature of the feasible sets X_i or of the functions Θ_i. For example, as we said, a game is *non-cooperative* if each player pursues his/her own interests working independently and without assuming anything about what other players are doing. In case the elements of the feasible sets X_i are functions of time, we are dealing with a *dynamic game*. A non dynamic game is *static*. Furthermore, if all the feasible sets X_i of a game Γ are finite sets, we say that Γ is a *finite game*.

A.3 Solution concepts: Nash Equilibrium and Generalized Nash Equilibrium

Given a game Γ, which strategies will the rational players adopt? In other words, which variable $x_i \in X_i$ will every player choose? We will call such a point a solution of the game. Intuitively, a player pursue the case in which his cost is the lowest possible. Since the payoff function Θ_i depends even on the strategies of the other players which in turn are minimizing their own costs, a conflict situation is created and it is not easy to characterize the best choice for every player. In other words, when rational players correctly forecast the strategies of their opponents they are not merely playing best responses to their beliefs about their opponents' play; they are playing best responses to the actual play of their opponents. Indeed, the notion of a solution is more tenuous in game theory than in other fields; it concerns with optimality, feasibility and equilibria.

In the fifties a solution concept - due to John Forbes Nash, see [71] - emerged as the most appropriate and effective. When all players correctly forecast their opponents' strategies, and play best responses to these forecasts, the resulting strategy profile is a Nash equilibrium. We will firstly introduce the Nash solution concept and then the generalizations of it.

Since we have to analyze the behavior of player i for a given and fixed choice of all the other players variables, let us denote with $\mathbf{x}_{-i}$ the set of all the players variables, except the i-nth one:

$$\mathbf{x}_{-i} \equiv (x_1, x_2, ..., x_{i-1}, x_{i+1}, ..., x_n)$$

so we can write $\mathbf{x} = (x_i, \mathbf{x}_{-i})$.

Definition 2. *A vector $\overline{\boldsymbol{x}} \in X$ is called a Nash equilibrium (NE) for the game if:*

$$\Theta_i(\overline{\boldsymbol{x}}) \leq \Theta_i(x_i, \overline{\boldsymbol{x}}_{-i}), \quad \forall x_i \in X_i$$

holds for all $i \in N$.

Equivalently, $\overline{\mathbf{x}}$ is a Nash equilibrium if and only if $\overline{x}_i$ solves the minimization problem:

$$\min_{x_i} \Theta_i \left(x_i, \overline{\mathbf{x}}_{-i}\right), \quad \text{s.t.} \quad x_i \in X_i$$

for all $i \in N$, i.e., if and only if no player can improve his objective function by *unilaterally* changing his strategy.

We note that the above definition neither implies that a strategic game has a Nash equilibrium, nor that it has only one. Indeed, some games have a single Nash equilibrium, some possess no Nash equilibria and others have many Nash equilibria.

The solution concept proposed by Nash implies that every player is playing a best response to the strategy choices of his/her opponents. Nevertheless, in some conflict situations (for example the one describing the behavior of SaaSs and IaaSs studied in this work, see Chapter 3) the strategy set of one player depends on the choices of the others. In particular, in a *Nash Equilibrium Problem (NEP)*, the feasible set X_i of the i-nth player depends only on his own variables. Differently, we define a *Generalized Nash Equilibrium Problem (GNEP)* when not only the objective functions of each player depend upon the strategies chosen by all the other players, but also each player's feasible set depends on the rival players' strategies, meaning that the constraints of the i-nth player may depend on all the decision variables of the game.

It is important to note that, in both cases, the objective functions Θ_i may depend on the whole set of variables.

Definition 3. *Assume that* $\overline{\boldsymbol{x}} = (\overline{x}_1, \overline{x}_2, ..., \overline{x}_n) \in X$ *is a feasible point for a given game* Γ. *The vector* $\overline{\boldsymbol{x}}$ *is called a Generalized Nash Equilibrium (GNE) for the game if:*

$$\Theta_i(\overline{\boldsymbol{x}}) \leq \Theta_i(x_i, \overline{\boldsymbol{x}}_{-i}), \quad \forall x_i \in X_i\left(\overline{\boldsymbol{x}}_{-i}\right) \tag{A.2}$$

holds for all $i \in N$.

Equivalently, $\overline{\mathbf{x}}$ is a Nash equilibrium if and only if $\overline{x}_i$ solves the optimization problem:

$$\min_{x_i} \Theta_i \left(x_i, \overline{\mathbf{x}}_{-i}\right), \quad \text{s.t.} \quad x_i \in X_i\left(\overline{\mathbf{x}}_{-i}\right)$$

for all $i \in N$, i.e., if no player can decrease his objective function by changing *unilaterally* his strategy $\overline{x}_i$ to any other feasible point.

Throughout, we assume that the GNEP satisfies the following assumptions:

- The objective functions Θ_i are continuous;
- The objective functions $\Theta_i(\cdot, \mathbf{x}_{-i})$ are convex as a mapping of x_i alone;
- The strategy set $X_i(\mathbf{x}_{-i})$ is defined explicitly by inequation constraints:

$$X_i(\mathbf{x}_{-i}) = \{x_i \in \mathbb{R}^{n_i} : \ g_i(x_i, \mathbf{x}_{-i}) \leq 0\}. \tag{A.3}$$

At the current state of the art a GNEP is much more difficult to solve than a standard NEP and only in few specific cases it is tractable. This is the reason why it is possible to define another class of generalized problems that are very common in practice: the Jointly-Convex GNEP (JC-GNEP). In literature, this type of problems is even called NEP with Shared Constraints.

Definition 4. *Assume that for every player i and every $\boldsymbol{x}_{-i}$, the objective function $\Theta_i(\cdot, \boldsymbol{x}_{-i})$ is convex and the set $X_i(\boldsymbol{x}_{-i})$ is closed and convex. A Jointly-Convex Generalized Nash Equilibrium Problem (JC-GNEP), or GNEP with Shared Constraints, is defined as the following optimization problem:*

$$\min_{x_i} \Theta_i(x_i, \overline{\boldsymbol{x}}_{-i}), \quad s.t. \quad x_i \in X_i(\boldsymbol{x}_{-i}) \tag{A.4}$$

where

$$X_i(\boldsymbol{x}_{-i}) = \{x_i \in \mathbb{R}^{n_i} : \ (x_i, \boldsymbol{x}_{-i}) \in X\} \tag{A.5}$$

for some closed convex set $X \subseteq \mathbb{R}^M$.

Again, when the sets $X_i(\mathbf{x}_{-i})$ are defined explicitly by a system of inequalities, then it is easy to check that (A.5) is equivalent to the requirement that $g_1 = g_2 = ... = g_N = g$ and that $g(\mathbf{x})$ be componentwise convex with respect to all variables $\mathbf{x}$. Furthermore, in this case, it obviously holds that:

$$X = \left\{\mathbf{x} \in \mathbb{R}^M : \ g(\mathbf{x}) \leq 0\right\}.$$

More generally, a JC-GNEP may be defined as:

$$\min_{x_i} \Theta_i(x_i, \mathbf{x}_{-i}), \tag{A.6}$$

$$\text{s.t.} \quad x_i \in X_i(\mathbf{x}_{-i}) = \{x_i \in \mathbb{R}^{n_i} : \ g_i(x_i) \leq 0\} \tag{A.7}$$

$$h(\mathbf{x}) = h(x_i, \mathbf{x}_{-i}) \leq 0 \tag{A.8}$$

where the constraints g_i of player i depend only on his own variables, while constraints common to all players, h, depend on the variables set. Sometimes the first constraints are called *private*, the latters *shared* or *common constraints*. Once again, we consider that the assumptions made for GNEP are still valid, adding the assumption that $h_i(\mathbf{x})$ are convex in respect to all variables.

A summary of the different classes of Nash problems and the relation among them is given in Figure A.1.

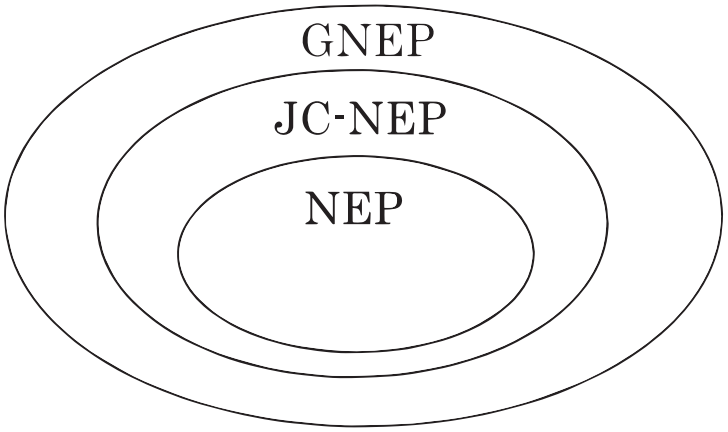

Figure A.1: Families of Generalized Nash Equilibrium Problems.

A.4 Equilibria existence and potential games

The Cloud Computing problem considered in this work can be studied through a game-theory methodology.

The aim of this section is to supply some basic results of the equilibria existence for the GNEP. In particular we focus on the definition of potential function for game Γ and illustrate what is a potential game.

Existence of solution for problem (A.2) has been the main focus of early research in GNEPs. The Debreu paper, [28], where the GNEP was formally introduced, also gives the first existence theorem. This existence result was based on fixed-point arguments, and this turned out to be the main proof tool used in the literature. The main existence result is probably the one established in Arrow and Debreu, [20]. We report below a slightly simplified version given by Ichiishi [51].

Theorem A.4.1. *Given a GNEP (A.2), assume that* $\Theta_i \in C^0(\mathbb{R}^{n_i})$ *for all* $i \in N$ *and suppose that:*

a) there exists n nonempty, convex and compact sets $K_i \subset \mathbb{R}^{n_i}$ *such that for every* $\boldsymbol{x} \in \mathbb{R}^N$ *with* $x_i \in K_i$ *for every i,* $X_i(\boldsymbol{x}_{-i})$ *is nonempty, closed and convex,* $X_i(\boldsymbol{x}_{-i}) \subseteq K_i$*, and* X_i*, as a point-to-set map, is both upper and lower semicontinuos;*

b) for every player i, the functions $\Theta_i(\cdot, \boldsymbol{x}_{-i})$ *is quasi-convex on* $X_i(\boldsymbol{x}_{-i})$*.*

Then a generalized Nash equilibrium exists.

Note that when the sets X_i are defined by inequalities constraints as in (A.3), the lower and upper semicontinuity requirements translate into reasonably mild conditions on the functions g_i.

For the most of the applied problems studied with a game-theory approach, the assumptions of the above theorem are too strong to be satisfied. Hence, to guarantee the existence of a solution, alternative results have to be considered. Some researches focus on the analysis of the assumptions, their relaxation or the formulation of weaker conditions. For example, the relaxation of the assumptions in the previous theorem has been the subject of an intense study [22], [70], [77], [89]. However, we have not discussed it in this work since it is not of interest for the problem considered.

Another way of overcoming the equilibria existence problem for applications where assumptions of Theorem A.4.1 are not guaranteed, is to identify potential functions that allow to formulate the so called potential game [69], [93], [33]. The first formal definition of potential games was given by Monderer and Shapley: in their work [69] they illustrate and discuss several notions of potential functions for games in strategic form, characterizing games that have a potential function and presenting a variety of applications.

In the remainder of this section we will provide an illustration of the main concepts of potential games. With the presented theory and tools we have been able to analyze the problem under study and provide an equilibria existence result (see Section 3.4).

A game is said to be a potential game if the incentive of all players to change their strategy can be expressed using a single function called the potential function. The potential function is a useful tool to analyze equilibrium properties of games, since the incentives of all players are mapped into one function, and the set of Nash equilibria can be found by locating optima of the potential function.

Definition 5. *A strategic game* $\Gamma = \left\{N, \{X_i\}_{i \in N}, \{\Theta_i\}_{i \in N}\right\}$ *is:*

- *an exact potential game if there exists a function* $\Pi : X \to \mathbb{R}$ *such that for all* $\boldsymbol{x}_{-i} \in X_{-i}$ *and for all* $y_i,\, z_i \in X_i$*:*

$$\Theta_i\left(y_i,\, \boldsymbol{x}_{-i}\right) - \Theta_i\left(z_i,\, \boldsymbol{x}_{-i}\right) = \Pi\left(y_i,\, \boldsymbol{x}_{-i}\right) - \Pi\left(z_i,\, \boldsymbol{x}_{-i}\right), \quad \forall i \in N; \tag{A.9}$$

- *an ordinal potential game if there exists a function* $\Pi : X \to \mathbb{R}$ *such that for all* $\boldsymbol{x}_{-i} \in X_{-i}$ *and for all* $y_i,\, z_i \in X_i$*:*

$$\Theta_i\left(y_i,\, \boldsymbol{x}_{-i}\right) - \Theta_i\left(z_i,\, \boldsymbol{x}_{-i}\right) > 0 \;\Leftrightarrow\; \Pi\left(y_i,\, \boldsymbol{x}_{-i}\right) - \Pi\left(z_i,\, \boldsymbol{x}_{-i}\right) > 0, \quad \forall i \in N; \tag{A.10}$$

Such a function Π *is called an (exact, ordinal or generalized) potential function of the game* Γ*.*

Clearly, an exact potential game is an ordinal potential game. In exact potential games the difference in the value of the potential equal the difference in the payoff to the deviating player. In ordinal potential games only the signs of the differences match.

The *potential minimizer* of an ordinal potential game Γ is the set of strategy combinations $\mathbf{x} \in X$ for which some potential Π achieves a maximum. The following proposition follows immediately from these definitions.

Property A.4.2. *Let* $\Gamma = \left\{N,\, \{X_i\}_{i \in N},\, \{\Theta_i\}_{i \in N}\right\}$ *be an ordinal potential game and* Π *a potential for* Γ*. If* $\boldsymbol{x} \in X$ *is a Nash equilibrium of* $\Gamma^{Pot} = \left\{N,\, \{X_i\}_{i \in N},\, \Pi\right\}$*, i.e., of the game with all payoff functions replaced by* Π*, then* $\boldsymbol{x}$ *is a Nash equilibrium of* Γ*. In particular, every ordinal potential game has at least one Nash equilibrium, since the potential minimizer is nonempty.*

We note that in case Γ is an exact or ordinal potential game and $\mathbf{x}$ is a Nash equilibrium of Γ, then $\mathbf{x}$ is also a Nash equilibrium of Γ^{Pot}.

The minimizers of Π on the set X are called *social equilibria* of the game. It is clear from the above definitions that each social equilibrium is a special NE, indeed no one player can improve its payoff by unilaterally deviating his strategy. In other words, social equilibria represent the NE which are optimal from a social point of view.

Roughly speaking a generalized potential game is a GNEP where the players are (unknowingly) minimizing the same function and where the feasible set of each player is the "section" of a larger nonempty set in the product space $\mathbb{R}^{\mathbb{M}}$. Hence, the two fundamental ingredients to evaluate if the game we are dealing with is a generalized potential game are:

- a common feasible set X;

- a potential function Π that reflects the changes in the players' objective functions.

In [36] the concept of potential game has been extended for GNEP with joint constraints. A generalization of the results of the classical definition of potential game is given considering the elimination of the assumptions of a Cartesian product structure for X and of the usual convexity either on the set X or on the objective functions $\Theta_i(\cdot, \mathbf{x}_i)$. Moreover, the author illustrates how to identify a potential function in the peculiar case in which the objective functions do not depend on the other players' variables: the potential function is simply given by the sum of the objective functions of all players (for an example of applications see also [17]); another common case considered is when $\Theta_i(\mathbf{x}) = c(\mathbf{x}) + d_i(x_i)$, that is when the objective functions have a common term c which is the same for all players plus an additional cost related only to x_i: then $\Pi(\mathbf{x}) = c(\mathbf{x}) + \sum_{i=1}^{N} d_i(x_i)$. Despite the clear analysis of such interesting cases, the authors just focus their attention on problems where the feasible set X is defined by some inequalities constraints:

$$X = \left\{\mathbf{x} \in \mathbb{R}^N : \ g(\mathbf{x}) \leq 0\right\},$$

pursuing a discussion on potential game where the constraints $g(\mathbf{x}) \leq 0$ are shared by all the players.

A.5 Wardrop equilibrium

Another important concept of equilibrium was developed by the English transport analyst John Glen Wardrop [96].

In studies about traffic assignment, network equilibrium models are commonly used for the prediction of traffic patterns in transportation and telecommunication networks that are subject to congestion. The idea of traffic equilibrium is related to the concept of Nash equilibrium but the analysis results more difficult since in transportation networks there are many players to consider.

This alternative concept assumes that players (travelers or packets) select a route that they perceive as being the shortest under the prevailing traffic conditions, minimizing the time or cost incurred in their traversal. Since, the situation resulting from these individual decisions is one in which drivers cannot reduce their journey times by unilaterally choosing another route, the resulting traffic pattern represents an equilibrium, known as the *Wardrop (or user) equilibrium*.

Wardrop stated two principles that formalize this notion of equilibrium and the alternative postulate of the minimization of the total travel costs. His first principle reads:

Property A.5.1 (Wardrop first principle)**.** *The journey times on all the routes actually used are equal, and less than those which would be experienced by a single vehicle on any unused route.*

Specifically, a user-optimized equilibrium is reached when no user may lower his transportation cost through unilateral action.

Property A.5.2 (Wardrop second principle)**.** *At equilibrium the average journey time is minimum.*

This implies that each user behaves cooperatively in choosing his own route to ensure the most efficient use of the whole system.

Wardrop's first principle of route choice became accepted as a sound and simple behavioral principle to describe the spreading of trips over alternate routes due to congested conditions. Its first mathematical formalization was introduced in the context of transportation networks by Beckmann, McGuire, and Winsten, in 1956. Transportation planners have been using Wardrop equilibrium models to predict computers decisions in real life networks.

Bibliography

[1] *2011 International Conference on Distributed Computing Systems, ICDCS 2011, Minneapolis, Minnesota, USA, June 20-24, 2011*. IEEE Computer Society, 2011.

[2] *INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10-15 April 2011, Shanghai, China*. IEEE, 2011.

[3] *2012 IEEE 32nd International Conference on Distributed Computing Systems, Macau, China, June 18-21, 2012*. IEEE, 2012.

[4] *26th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IPDPS 2012, Shanghai, China, May 21-25, 2012*. IEEE Computer Society, 2012.

[5] Vineet Abhishek, Ian A. Kash, and Peter Key. Fixed and market pricing for cloud services. *CoRR*, abs/1201.5621, 2012.

[6] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review.

[7] Jussara Almeida, Virgílio Almeida, Danilo Ardagna, ítalo Cunha, Chiara Francalanci, and Marco Trubian. Joint admission control and resource allocation in virtualized servers. *J. Parallel Distrib. Comput.*, 70(4):344–362, April 2010.

[8] Eitan Altman, Thomas Boulogne, Rachid El Azouzi, Tania Jiménez, and Laura Wynter. A survey on networking games in telecommunications. *Computers & OR*, 33:286–311, 2006.

[9] Amazon Inc. Amazon Elastic Cloud Computing. http://aws.amazon.com/ec2/.

[10] Amazon Inc. Amazon Web Services. http://aws.amazon.com/.

[11] AMPL. Ampl modeling language for mathematical programming. http://www.ampl.com/.

[12] Apache. Apache JMeter. http://jmeter.apache.org/.

[13] D. Ardagna, E. di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D"Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu, A. Gericke, and C. Sheridan. Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. In *Modeling in Software Engineering (MISE), 2012 ICSE Workshop on*, pages 50–56, 2012.

[14] Danilo Ardagna, Sara Casolari, Michele Colajanni, and Barbara Panicucci. Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems. *J. Parallel Distrib. Comput.*, 72(6):796–808, 2012.

[15] Danilo Ardagna, Sara Casolari, and Barbara Panicucci. Flexible distributed capacity allocation and load redirect algorithms for cloud systems. In Liu and Parashar [58], pages 163–170.

[16] Danilo Ardagna, Barbara Panicucci, and Mauro Passacantando. A game theoretic formulation of the service provisioning problem in cloud systems. In Srinivasan et al. [85], pages 177–186.

[17] Danilo Ardagna, Barbara Panicucci, and Mauro Passacantando. Generalized nash equilibria for the service provisioning problem in cloud systems. *Services Computing, IEEE Transactions on*, PP(99):1, 2012.

[18] Danilo Ardagna, Barbara Panicucci, Marco Trubian, and Li Zhang. Energy-aware autonomic resource allocation in multitier virtualized environments. *IEEE T. Services Computing*, 5(1):2–19, 2012.

[19] Michael Armbrust, Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, and Matei" Zaharia. Above the clouds: A berkeley view of cloud computing. 2009.

[20] Kenneth Arrow and Gerard Debreu. Existence of an equilibrium for a competitive economy. *Econometrica*, 22(3):265–290, 1954.

[21] Jeff Barr. *Host Your Web Site In The Cloud: Amazon Web Services Made Easy Amazon EC2 Made Easy*. Sitepoint, 1st edition, 2010.

[22] Michael R Baye, Guoqiang Tian, and Jianxin Zhou. Characterizations of the existence of equilibria in games with discontinuous and non-quasiconcave payoffs. *Review of Economic Studies*, 60(4):935–48, October 1993.

[23] R. Birke, L.Y. Chen, and E. Smirni. Data centers in the cloud: A large scale performance study. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 336–343, June.

[24] Mathias Björkqvist, Lydia Y. Chen, and Walter Binder. Opportunistic service provisioning in the cloud. In Chang [26], pages 237–244.

[25] Marco Caldirola. Tecniche di resource allocation per sistemi virtualizzati di larga scala. Master's thesis, Politecnico di Milano, 2010.

[26] Rong Chang, editor. *2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, June 24-29, 2012*. IEEE, 2012.

[27] L. Cherkasova and P. Phaal. Session-based admission control: a mechanism for peak load management of commercial web sites. *Computers, IEEE Transactions on*, 51(6):669–685, Jun.

[28] Gerard Debreu. A social equilibrium existence theorem. *Nat. Acad. Science*, 38:886–893, 1952.

[29] Brian Dougherty, Jules White, and Douglas C. Schmidt. Model-driven auto-scaling of green cloud computing infrastructure. *Future Generation Comp. Syst.*, 28(2):371–378, 2012.

[30] Parijat Dube, Zhen Liu, Laura Wynter, and Cathy H. Xia. Competitive equilibrium in e-commerce: Pricing and outsourcing. *Computers & OR*, 34(12):3541–3559, 2007.

[31] Parijat Dube, Corinne Touati, and Laura Wynter. Capacity planning, quality of service and price wars. *SIGMETRICS Performance Evaluation Review*, 35(3):31–33, 2007.

[32] Sourav Dutta, Sankalp Gera, Akshat Verma, and Balaji Viswanathan. Smartscale: Automatic application scaling in enterprise clouds. In Chang [26], pages 221–228.

[33] Yu.M. Ermoliev and S.D. Flaam. Repeated play of potential games. *Cybernetics and Systems Analysis*, 38:355–367, 2002.

[34] Francisco Facchinei, Andreas Fischer, and Veronica Piccialli. Generalized nash equilibrium problems and newton methods. *Math. Program.*, 117(1-2):163–194, 2009.

[35] Francisco Facchinei and Christian Kanzow. Generalized nash equilibrium problems. *Annals OR*, 175(1):177–211, 2010.

[36] Francisco Facchinei, Veronica Piccialli, and Marco Sciandrone. Decomposition algorithms for generalized potential games. *Computational Optimization and Applications*, 50:237–262, 2011.

[37] Yuan Feng, Baochun Li, and Bo Li. Price competition in an oligopoly cloud market. 2011.

[38] Flexyscale. http://www.flexiscale.com/.

[39] GoGrid. http://www.gogrid.com/.

[40] Google Inc. http://www.google.com/about/company/.

[41] Google Inc. Google App Engine. https://developers.google.com/appengine/.

[42] Google Inc. Google Apps for Business. http://www.google.com/enterprise/apps/business/.

[43] Google Inc. Google Compute Engine. https://cloud.google.com/products/compute-engine.

[44] Hadi Goudarzi and Massoud Pedram. Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems. In Liu and Parashar [58], pages 324–331.

[45] Albert G. Greenberg and Kazem Sohraby, editors. *Proceedings of the IEEE INFOCOM 2012, Orlando, FL, USA, March 25-30, 2012*. IEEE, 2012.

[46] Makhlouf Hadji and Djamal Zeghlache. Minimum cost maximum flow algorithm for dynamic resource allocation in clouds. In Chang [26], pages 876–882.

[47] Mohammad Mehedi Hassan, M.Shamim Hossain, A.M.Jehad Sarkar, and Eui-Nam Huh. Cooperative game-based distributed resource allocation in horizontal dynamic cloud federation platform. *Information Systems Frontiers*, pages 1–20, 2012.

[48] Mohammad Mehedi Hassan, Biao Song, and Eui nam Huh. Distributed resource allocation games in horizontal dynamic cloud federation platform. In Thulasiraman et al. [87], pages 822–827.

[49] Ting He, Shiyao Chen, Hyoil Kim, Lang Tong, and Kang-Won Lee. Scheduling parallel tasks onto opportunistically available cloud resources. In Chang [26], pages 180–187.

[50] IBM. IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

[51] Tatsuro Ichiishi. *Game Theory for Economic Analysis*. New York: Academic Press, 1983.

[52] Ganesh Neelakanta Iyer and Bharadwaj Veeravalli. On the resource allocation and pricing strategies in compute clouds using bargaining approaches. In Veeravalli and Foster [91], pages 147–152.

[53] JMeter Plugins. https://code.google.com/p/jmeter-plugins/.

[54] Kernel Based Virtual Machine. http://www.linux-kvm.org/.

[55] Kleopatra Konstanteli, Tommaso Cucinotta, Konstantinos Psychas, and Theodora A. Varvarigou. Admission control for elastic cloud services. In Chang [26], pages 41–48.

[56] Dara Kusic, Jeffrey O. Kephart, James E. Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

[57] Yi-Kuei Lin and Ping-Chen Chang. Reliability evaluation of a computer network in cloud computing environment subject to maintenance budget. *Applied Mathematics and Computation*, 219(8):3893–3902, 2012.

[58] Ling Liu and Manish Parashar, editors. *IEEE International Conference on Cloud Computing, CLOUD 2011, Washington, DC, USA, 4-9 July, 2011*. IEEE, 2011.

[59] Tieming Liu, Chinnatat Methapatara, and Laura Wynter. Revenue management model for on-demand it services. *European Journal of Operational Research*, 207(1):401–408, 2010.

[60] Markov4JMeter. http://se.informatik.uni-kiel.de/markov4jmeter/.

[61] Michele Mazzucco and Dmytro Dyachuk. Optimizing cloud providers revenues via energy efficient server allocation. *Sustainable Computing: Informatics and Systems*, 2(1):1 – 12, 2012.

[62] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing. Technical report, July 2009.

[63] Ishai Menache, Asuman Ozdaglar, and Nahum Shimkin. Socially optimal pricing of cloud computing resources. In *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '11, pages 322–331, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[64] Microsoft Corporation. http://www.microsoft.com/.

[65] Microsoft Corporation. Microsoft Office 365. http://office365.microsoft.com/.

[66] Microsoft Corporation. Microsoft Windows Azure. http://www.windowsazure.com/.

[67] Microsoft Corporation. Microsoft Windows Azure Virtual Machines. http://www.windowsazure.com/en-us/home/features/virtual-machines/.

[68] MODAClouds. http://www.modaclouds.eu/.

[69] Dov Monderer and Lloyd S. Shapley. Potential games. *Games and Economic Behavior*, 14(1):124–143, May 1996.

[70] Jacqueline Morgan and Vincenzo Scalzo. Pseudocontinuous functions and existence of nash equilibria. *Journal of Mathematical Economics*, 43(2):174 – 183, 2007.

[71] John Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, 1951.

[72] Elisabetta Di Nitto and Ramin Yahyapour, editors. *Towards a Service-Based Internet - Third European Conference, ServiceWave 2010, Ghent, Belgium, December 13-15, 2010. Proceedings*, volume 6481 of *Lecture Notes in Computer Science*. Springer, 2010.

[73] C.G.A.M. van den Nouweland, Peter Borm, W. van Golstein Brouwers, R. Groot Bruinderink, and S.H. Tijs. A game theoretic approach to problems in telecommunication. Open access publications from tilburg university, Tilburg University, 1996.

[74] Onlive. http://www.onlive.com/.

[75] Rackspace Inc. http://www.rackspace.com/.

[76] N.S.V. Rao, S.W. Poole, Fei He, Jun Zhuang, C.Y.T. Ma, and D.K.Y. Yau. Cloud computing infrastructure robustness: A game theory approach. In *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pages 34 –38, 30 2012-feb. 2 2012.

[77] Philip J. Reny. On the existence of pure and mixed strategy nash equilibria in discontinuous games. *Econometrica*, 67(5):1029–1056, 1999.

[78] Salesforce Inc. http://www.force.com/.

[79] SAP. http://www.sap.com/.

[80] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A cost-aware elasticity provisioning system for the cloud. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 559 –570, june 2011.

[81] Upendra Sharma, Prashant J. Shenoy, Sambit Sahu, and Anees Shaikh. Kingfisher: Cost-aware elasticity in the cloud. In *INFOCOM* [2], pages 206–210.

[82] Socialbakers. Facebook statistics. http://www.socialbakers.com/facebook-statistics/.

[83] Yang Song, Murtaza Zafer, and Kang-Won Lee. Optimal bidding in spot instance market. In Greenberg and Sohraby [45], pages 190–198.

[84] Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pages 10–10, Berkeley, CA, USA, 2008. USENIX Association.

[85] Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors. *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*. ACM, 2011.

[86] Standard Performance Evaluation Corporation. SPECweb2005. http://www.spec.org/web2005/.

[87] Parimala Thulasiraman, Laurence Tianruo Yang, Qiwen Pan, Xingang Liu, Yaw-Chung Chen, Yo-Ping Huang, Lin-Huang Chang, Che-Lun Hung, Che-Rung Lee, Justin Y. Shi, and Ying Zhang, editors. *13th IEEE International Conference on High Performance Computing & Communication, HPCC 2011, Banff, Alberta, Canada, September 2-4, 2011*. IEEE, 2011.

[88] Fengguang Tian and Keke Chen. Towards optimal resource provisioning for running mapreduce programs in public clouds. In Liu and Parashar [58], pages 155–162.

[89] Guoqiang Tian and Jianxin Zhou. Transfer continuities, generalizations of the weierstrass and maximum theorems: A full characterization. *Journal of Mathematical Economics*, 24(3):281 – 303, 1995.

[90] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, December 2008.

[91] Bharadwaj Veeravalli and Ian T. Foster, editors. *Proceedings of the 17th IEEE International Conference on Networks, ICON 2011, Singapore, December 14-16, 2011*. IEEE, 2011.

[92] VMware Inc. http://www.vmware.com/.

[93] Mark Voorneveld. Equilibria and approximate equilibria in infinite potential games. *Economics Letters*, 56(2):163 – 169, 1997.

[94] Jian Wan, Dechuan Deng, and Congfeng Jiang. Non-cooperative gaming and bidding model based resource allocation in virtual machine environment. In *IPDPS Workshops* [4], pages 2183–2188.

[95] Hongyi Wang, Qingfeng Jing, Rishan Chen, Bingsheng He, Zhengping Qian, and Lidong Zhou. Distributed systems meet economics: pricing in the cloud. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, pages 6–6, Berkeley, CA, USA, 2010. USENIX Association.

[96] J Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers, Part II*, 1(36):352–362, 1952.

[97] Guiyi Wei, Athanasios V. Vasilakos, Yao Zheng, and Naixue Xiong. A game-theoretic method of fair resource allocation for cloud computing services. *The Journal of Supercomputing*, 54(2):252–269, 2010.

[98] Andreas Wolke and Gerhard Meixner. Twospot: A cloud platform for scaling out web applications dynamically. In Nitto and Yahyapour [72], pages 13–24.

[99] Linlin Wu, Saurabh Kumar Garg, and Rajkumar Buyya. Sla-based admission control for a software-as-a-service provider in cloud computing environments. *J. Comput. Syst. Sci.*, 78(5):1280–1299, 2012.

[100] Xen. Xen Hypervisor. http://www.xen.org/.

[101] Z. Xiao, Q. Chen, and H. Luo. Automatic scaling of internet applications for cloud computing services. *Computers, IEEE Transactions on*, PP(99):1, 2012.

[102] Z. Xiao, W. Song, and Q. Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *Parallel and Distributed Systems, IEEE Transactions on*, PP(99):1, 2012.

[103] PengCheng Xiong, Zhikui Wang, Simon Malkowski, Qingyang Wang, Deepal Jayasinghe, and Calton Pu. Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach. In *ICDCS* [1], pages 571–580.

[104] Murtaza Zafer, Yang Song, and Kang-Won Lee. Optimal bids for spot vms in a cloud for deadline constrained jobs. In Chang [26], pages 75–82.

[105] Sharrukh Zaman and Daniel Grosu. An online mechanism for dynamic vm provisioning and allocation in clouds. In Chang [26], pages 253–260.

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

[107] Qi Zhang, Quanyan Zhu, Mohamed Faten Zhani, and Raouf Boutaba. Dynamic service placement in geographically distributed clouds. In *ICDCS* [3], pages 526–535.

[108] Xiaoyun Zhu, Donald Young, Brian J. Watson, Zhikui Wang, Jerry Rolia, Sharad Singhal, Bret McKee, Chris Hyser, Daniel Gmach, Rob Gardner, Tom Christian, and Ludmila Cherkasova. 1000 islands: an integrated approach to resource management for virtualized data centers. *Cluster Computing*, 12(1):45–57, 2009.