Universidade do Minho
Escola de Engenharia

Bruno Luís da Silva e Costa

Dependable Cloud Computing Management Services

Outubro de 2010

Universidade do Minho
Escola de Engenharia

Bruno Luís da Silva e Costa

Dependable Cloud Computing Management Services

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efectuado sob a orientação do
Prof. António Luís Sousa

Outubro de 2010

Acknowledgements

To my advisor Prof. António Sousa, for his invaluable assistance and all the guidance during this project. Deepest gratitude also to members of Distributed Systems Group, Prof. Rui Oliveira and Prof. José Pereira for the critics and support.

To Miguel Matos, my teammate at DC2MS project, for all his vision and invaluable assistance. Special thanks to all my colleagues at distributed systems laboratory for providing me an excellent working environment and for their friendship. To all the department members and colleagues, with a special word of gratitude to my group members.

To Liliana, for all her love and patience.

To my beloved family and friends, for their understanding and patience through the duration of this work.

Resumo

A Computação em Nuvem tem vindo a ganhar um papel preponderante nas Tecnologias de Informação nos últimos anos, impulsionado pelo *boom* da Web nos finais do século XX. A visão de proporcionar ao utilizador recursos informáticos como um serviço é assim um sonho tornado realidade. Contudo, para manter estes sistemas com um elevado nível de confiabilidade, os serviços de gestão devem ser inerentemente escaláveis e tolerantes a faltas, para proporcionarem aos seus utilizadores um serviço ininterrupto e que dê as garantias necessárias para a adopção dos serviços da Computação em Nuvem, sem restrições.

Apesar da natureza crítica destes serviços de gestão, não existem soluções em sistemas reais que satisfaçam os requisitos de confiabilidade exigidos para uma adopção da Computação em Nuvem, nomeadamente soluções de gestão em que existam vários *data-centers* envolvidos. Sendo assim, este trabalho centra-se no desenho, análise e implementação de diversas arquitecturas para os serviços de gestão da Computação em Nuvem.

Abstract

Cloud Computing has become an important paradigm in the IT field, following the Web boom at the turn of the century. As such, the long-held dream of computing resources as a service became true. However, to keep these systems with a high level of resilience and reliability, the Cloud management services must be inherently scalable and fault tolerant, delivering to its users a dependable service, thus ensuring an accelerated adoption of Cloud Computing.

Even though these services are one of the critical components in the Cloud Computing scenario, there are no management service solutions whose dependability satisfies the requirements to a trustworthy adoption of the Cloud Computing paradigm, namely those which involves multi data-center Cloud infrastructures. As such, this work focus on the design, analysis and implementation of different architectures proposed for management services in Cloud Computing.

Contents

List of Figures

List of Acronyms

API Application Programming Interface

AWS Amazon Web Services

CAD Computer-aided Design

CC Cloud Computing

CPU Central Processing Unit

CRM Customer Relationship Management

CRUD Create, Read, Update and Delete

DBMS Database Management System

EC2 Elastic Cloud Computing

GCS Group Communication Service

IaaS Infrastructure as a Service

IT Information Technology

JPA Java Persistence API

LAN Local Area Network

MS Management Server

NIC Network Interface Card

PaaS Platform as a Service

PDF Portable Document Format

REST Representational State Transfer

SaaS Software as a Service

SLA Service Level Agreement

SOAP Simple Object Access Protocol

TCO Total Cost of Ownership

VLAN Virtual LAN

VNET Virtual Network

WN Worker Node

WYSIWYG What You See Is What You Get

XML Extensible Markup Language

Chapter 1

Introduction

The IT started long ago, in a time where computing was concentrated on main-frames on large-sized rooms. By that time, computers were expensive and bulky, having its users submitting its job one at a time. Thus, such powerful units were, inherently, time-shared systems, used in batch mode to maximize their use and profit. By that era (1943), the idea of distributed computing was not on the thoughts of big enterprises, as some important people as Thomas J. Watson (by the time president of IBM) stated "I think there is a world market for about five computers".

Later, the democratization of computers lead to easy access to computing at our homes. The personal computer era had begun. Computers were size-reduced and made available to the consumer market. The hype was that every desktop and laptop had sufficient capabilities and processing power to meet most of everyday tasks, such as office productivity tools and other CAD/design tasks. The computers became accessible to everyone, gaining exponential power across the years.

The network revolution brought the democratization of the Web, which made the enterprises look to the Web as a new market to promote their products and services. This so called dot-com boom in the mid-1990s revealed data-centers with highly concentrated power and reduced size to meet the Web requirements. Being considered as a priority, the Web services became the core show-room for many enterprises. This exclusively priced data-centers were only available to enterprise-scale customers that justify this up-front commitment.

As a result, most medium to large-sized organizations invest on their data-center to provide customers with their services. However, this infrastructure needs to be over-provisioned to sustain peak loads,

and most of the time only a fraction of the resources were used.

Deploying these services on an elastic service will help the enterprises to pay only for the resources needed. For all of this, the emergence Cloud Computing paradigm.

1.1 Motivation

A Cloud Computing environment comprises a collection of nodes, that are served to customers, using different abstractions. The interface to the Cloud needs to be transparent and oblivious to the architecture of the physical infrastructure, providing a trustworthy service.

The core motivation of this work is to study the mechanisms used in the management of a Cloud infrastructure, searching for architectures that provide a dependable management service, which will greatly improve the adoption of the Cloud. Studying these mechanisms involves delving into inner details of data-center operations, revisiting typical infrastructure best-practices, and finally improving them on the new perspective of the Cloud.

Current cloud management solutions do not account for a dependable and scalable management service, and tend to rely on centralized architectures in which one controller has god-like vision of the whole system. On one hand, a fault at the controller compromises the dependability, being a single point of failure. On the other hand, the scalability is limited by the resources of the controller.

To overcome this limitation, new approaches need to be developed, that assures scalability, reliability and dependability at a data-center scenario. This involves the use of replication, synchronization and coordination mechanisms large infrastructures as in a typical data-center scenario.

This management service of the Cloud Computing environment is the core of this study. In order to build a dependable Cloud management service, resource information needs to be replicated and synchronized over a large number of nodes.

These motivations were also recognized by the IT community, namely in the HPLabs DC2MS [2] project, which supported this study.

1.2 Objectives

As outlined in the previous Section, a management service to a Cloud environment needs to carefully address aspects of dependability, scalability and reliability. In order to provide these guarantees, an architecture that fulfills these requirements is fundamental. Unfortunately, as current solutions are driven by centralized-based architectures, it becomes the main goal of this work to design new architectures that provide a dependable and scalable solution to a Cloud Computing management service.

Studying these architectures we want to be able to clearly state the advantages and disadvantages, and fundamentally have insight on the better architecture for a giving scenario. This comes as priority because nowadays Cloud Computing infrastructures involves multiple location deployments, and we should reason on the usefulness of a solution in a concrete deployment.

Following this study, the solutions purported need to be throughly evaluated, using a meaningful scenario in the context of Cloud Computing. This involves evaluation of the architectures considering a large-scale deployment.

1.3 Dissertation Outline

The rest of the dissertation is organized as follows. Chapter 2 brings an introduction to the Cloud Computing paradigm, giving an overview of different types of services offered in the market. Afterwards, different aspects of the Cloud are studied, focusing on the management services for the Cloud. Chapter 3 presents the problem that is being addressed, stating the rationale behind it. Chapter 4 carefully describes the different solutions proposed, discussing the advantages and disadvantages of each proposal. Chapter 5 evaluates the architecture in the light of the requirements presented at Chapter 3. And finally, Chapter 6 concludes the dissertation, presenting the results obtained from this work, and pointing directions towards future research in this subject.

Chapter 2

Related Work

This Chapter gives an overview of the current technologies related to Cloud Computing. It starts by giving an overall vision of the Cloud Computing as an emerging platform, and then gradually converges to the internal details.

2.1 Background

2.1.1 Cloud Computing Defined

Cloud Computing appears as "the long-held dream of computing as a utility" [7] in which developers have a platform to deploy their services on demand. In other words, Cloud Computing provides to the IT industry a way to move their infrastructure, applications and services to remotely managed servers. These servers usually provide ways of monitoring and acquaint for the Cloud state, but in the majority of the cases it is the responsibility of the providers to account for the condition of the services. The burden of administration is on Cloud provider's side, which brings the focus to development rather than deployment, configuration and management.

From the point of view of the developer, the Cloud reliefs him/her from the complex infrastructure planning, that become unpredictable when there is unawareness related to the success of a project. For start-ups and even some experimental projects, it is difficult to account for resources in advance: the deployment of an application implies having the human resources with the required skills and having access to an hardware platform to deploy the services.

Having justified the need for computing as an utility, a more clear

definition of Cloud Computing is still missing. In [47], the authors try to reach a Cloud Computing definition, since the concept is not itself consensual in the community. The Cloud, as conceived today, is a large pool of virtualized resources (computing, storage and network) that are made available to the public. These resources are dynamically provisioned in a way that meets usage demand, being able to scale up and down appropriately. The model of usage is in a pay-per-use fashion, where expenditures apply only to resources being used.

A key element of Cloud Computing is "the illusion of infinite resources" [7]. Being able to add and remove resources on demand eliminates the need of capacity planning for provisioning. It also enables Cloud users to provision more resources during peak loads, and stay at minimum resources on the other periods. Planning the hardware resources is not a trivial task since the usage and popularity of a world-wide service is not predictable.

Moreover, the resources should appear to the end-user isolated from other clients. Thus, the infrastructure should be able to appear to the consumer as if he/she is the only one to access resources, despite of other tenants. The disruption of nearby machines should leave the resources from a client unaffected. Another key aspect is the reduced time-to-market the Cloud provides for most business, as it is imperative to deploy the system earliest as possible. Developing having the Cloud in mind, means that Web platforms can be developed rapidly, and have scalability concerns from the beginning.

The dynamic provisioning provided by the Cloud also means that resources can be used and released at will. This empowers the enterprises with a mechanism to do batch jobs quickly, without an investment on large number of hardware components. A well-known study case was the use of the Cloud by the Washington Post to quickly deliver scanned PDFs to searchable online texts [4]. Being the sources with 17,481 pages length, it would take approximately 30 minutes per page to deliver a single page to the readers. Having the possibility to use the Cloud, Washington Post used 200 instances at Amazon EC2 to prepare the documents to be delivered in only 26 hours.

2.1.2 Cloud versus Grid

A comparison between Cloud Computing and Grid is an inevitable stop for those who first contact with the concept of Cloud Comput-

ing. The perception of the Grid Computing and Cloud Computing is that they both take advantage of technological advances such as multi-core processors and networked computing environments to deliver the customers new services that were not achievable with individual computers [12]. However, a more careful examination of the two topics clearly identify the differences between them.

In Grid Computing, we have an aggregation of distributed resources (usually geographically dispersed and ruled by different organizations) that work together to achieve a result of a scientific application. The Grid works on a time-shared basis, for computations that require more than a single computer. On the other hand, Cloud Computing is based on virtualization of resources to allow several customers to use the same infrastructure (a property known as multi-tenancy). The Cloud should be able to capture and solve scalability and reliability requirements whereas in Grid each organization must take account for their own.

2.1.3 Cloud on the Enterprise

Cloud Computing is very attractive to startup companies, where new services can be built quickly with a relatively low financial risk as it is evident that hardware equipment is not affordable by those whose success of the enterprise is not certain. Being a fixed cost, in-house deployment costs may not be sufficient to cover the income in an initial phase. However, for settled companies, the arguments in favor to the move to the Cloud simply do not apply. Established companies already have investment plans on equipment and can easily predict provisioning demands following the trends of the market and their expectations of growth of decay. Furthermore, enterprises have some other challenges on moving to the Cloud: "Security, legislation and dependence on the provider" [35] are just a few examples of problems that an enterprise needs to face. More challenges arise when customers have low bandwidth to the outside and need to move large amount of data into the Cloud.

While most of these problems yield true to the majority of the enterprises, in some scenarios it is still advantageous to use remote resources. The most evident is to accommodate peak loads. The idea is to have the necessary hardware resources in-house to normal conditions, expanding to the Cloud when the demand rises. Otherwise, these companies would need to be over-provisioned to maintain their services.

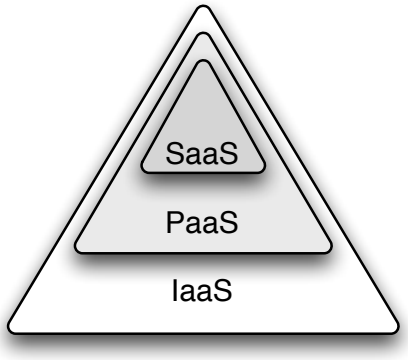

Figure 2.1: Cloud services levels.

The world conjecture in the latest years poses new questions in terms of recovery and fail-over mechanisms. Being able to move services to the Cloud in a catastrophic scenario is a good backup plan for those enterprises whose on-line services are a priority.

However, "Cloud Computing does not meet enterprise requirements yet" [45]. Having systems with infrastructures at multiple locations poses problems of low connectivity, flexibility of having information on both structures, among others.

New tools are required for enterprises to move transparently and automatically the services to the Cloud.

2.2 Cloud Computing Layers

The current Cloud Computing providers offer services to different usage scenarios. Analyzing these services, these can be categorized using a layered division focusing on the type of service provided. The services range from Web powered applications to full bare-metal virtual machines, passing by intermediate platforms in which users can deploy their applications. Based on these scenarios, some authors such as [47] and [19] agreed on this classification of Cloud layers: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).

The different abstraction levels of the Cloud stack can be built using the functionalities of the level below it, as can be seen in Figure 2.1.

2.2.1 Infrastructure as a Service

In the lowest abstraction level, the IaaS is offered as virtual machines to the end-users. These machines are assigned to each user on demand, from a previously selected range of operating systems. From the kernel up, everything in the stack is customizable. In this scenario, each user can define the organization of the services, the storage and the processing capabilities.

Despite of the responsibility of maintaining virtual machines up and running, the burden of administration is still shared with the developers, whom have to take care of operating system customization, application deployment and system tweaking. As such, scalability and dynamic provisioning can be done on a manual fashion, according to the demand of the users. We are able to define clear policies to choose when to scale-out and when to reduce resources provisioned, not being tied to the Cloud provider's rules.

One of the major players on this kind is Amazon EC2 [5]. Being one of the pioneers in this area, Amazon Web Services provides its users with virtual machines that can be deployed on demand. These machines are predefined in bundles that can be deployed using different configurations available. There are other providers that can be chosen, targeting the diversity of the market, such as GoGrid [21], Joyent [27], FlexiScale [16] and Rackspace Cloud [41] .

2.2.2 Platform as a Service

In the middle layer of the Figure 2.1, the PaaS support an abstraction for running applications in the Cloud. As the name implies, a platform is provided to the developers, in which they can build their applications. These platforms are usually locked-in, with APIs available to different languages. The development process is done on simulators/emulators before deploying the application into the Cloud. PaaS providers provide more high level services to the user, such as object storages and messaging services to help build scalable applications.

At this level of abstraction, the detail is usually smaller than in the IaaS. While some architectural aspects can be defined, there is no such detail as a concrete hardware definition. Multi-tenancy is also frequent in terms of application because applications run all on the same environment, whereas in IaaS applications runs on different virtual machines.

The major players in the IT industry have a strong presence in this type of Cloud Computing, mainly because PaaS leads to a better utilization of physical resources. This is due to the fact that many applications can be run on parallel in the same machine. Examples on this kind of Cloud are Google App Engine [23], Windows Azure Platform [34] and Force.com [43].

2.2.3 Software as a Service

At the top of the stack there is the SaaS. This comprises services that are ready in the Cloud to end-users. Many enterprises use these services to leverage the burden of developing applications. Typically this includes general services that are already well implemented such as e-mail, office, CRMs and calendars.

These services come as a real alternative to running local applications in individual computers. Office applications are a good example, as a small to mid-size company benefit from using a SaaS rather than acquiring licenses for the software. Additionally, the company benefits from a solution that is accessible to the customer via a Web browser, leading to a strong acceptance.

This less featured level constitutes most of the services on the Web. As an example, we can cite Yahoo! Mail [50] and Google Docs [24].

2.2.4 Other Classification Criteria

While this layered approach for Cloud Computing seems to be the most reasonable way to categorize the Cloud, other classifications can be done. Focusing on the location of deployment of the Cloud, we can have several types of Clouds: Public, Hybrid and Private Clouds.

Public Clouds are the traditional approach in the Cloud, and from whom the most benefits to the Cloud are obtained. These services are available to the public in general, using a contract and service level agreements. While this type of service seems to follow a traditional supply-demand model, once an application/service is acquired, there is a commitment between the user and the Cloud provider. End users are locked using one provider and moving to other provider later might not be an option because of the interface/feature mismatch. Having a Cloud provider following standards may improve the freedom of the user, but nowadays that is not a reality.

The ideas of Cloud Computing are also being incorporated in internal enterprise infrastructure. While many authors argue that the economical model of private cloud is nonsense, there are some advantages of running a private cloud. The concept is that services in an enterprise have different peak loads unsynchronized and the dynamic provisioning properties of the Cloud can be used.

Grids have always played an important role on academic environments. The Cloud Computing shares "the same original vision of grid computing" [13], in which computers are aggregated to form a superior infrastructure of computing. Thus, architectures and technologies involved tend to be similar, namely in terms of infrastructure management. In academic environments, a private Cloud environment can be an added value, where test environments can be setup rapidly and in an automatic fashion.

2.3 An Inside Perspective of the Cloud

Cloud Computing is typically seen from the point of view of a customer. However, an inside perspective comes more important when we delve into Cloud Computing details. Though, offering Computing as an Utility means that core features need to be provided [28]: 1) Unified Control - a single entry point to the Cloud; 2) Freedom from Physical Configuration - prevent rewiring of hardware on system reconfiguration; 3) Resource Sharing - multi-tenancy on the resources; 4) Resource Isolation - user should only be aware of his own resources. To fulfill this task, several pieces of a puzzle must fit together to provide the overall Cloud service. Here we are focusing mainly on IaaS Cloud Computing layer, because this type is the base to solving the others, as upper levels can be built upon it.

The core components inside inside the Cloud can be divided in three different categories: Server Virtualization, Network Virtualization and Storage Virtualization. These three aspects are carefully analyzed in detail in the next sections.

2.3.1 Server Virtualization

In a typical scenario of an IaaS, the Cloud presents an abstraction of a virtual machine to the end-users. Each virtual machine is isolated from the other machines in the Cloud, being failure independent from other machines. From the inside of the data-center, the system

administrators manage a pool of servers available to the outside, each with several virtual machines.

A Cloud infrastructure needs a pool of servers that are correctly synchronized to provide Cloud services. This type of synchronism is needed to account for physical machines availability, status and monitoring. The Cloud platforms need to be highly autonomic to reduce Total Cost of Ownership (TCO), meaning that most of the tasks are automated: "launching servers, shutting them down, load balancing, failure detection and handling" [10]. For this task, there is a thin layer of software that runs each node from the pool of servers to control the physical machine from the outside.

This thin layer includes an Application Programming Interface (API) to create, run, update and destroy virtual machines. The managed resources on a node include CPU slots, memory, network interfaces, network bandwidth, storage access and I/O bandwidth. This API is running on the host of the node from the pool, and is called Virtual Machine Manager (VMM). Several aspects of the Cloud are also dealt within the VMM, such as virtual machine migration, to provide a way of moving machines to provide elasticity, and to perform maintenance tasks.

This pool of servers must be administered autonomically using a Management Server that acquaints for the state of all the resources, probing for their state and providing monitoring facilities. With this management server we achieve an unified vision and control of the Cloud.

2.3.2 Network Virtualization

Another component of critical importance is the network. There are two key issues concerning these type of resources: 1) guarantee the isolation of the virtual machines in a client basis; 2) account for the performance of the network, reducing the network dedicated to the infrastructure; 3) maximize bandwidth to customers and account for fault tolerance.

In a typical environment, isolation is accomplished by deploying physical Virtual LANs (VLANs)s to the network. However, in a typical Cloud scenario the number of users easily breaks the 2^{12} limitation barrier of VLANs [1], and need hardware support from the routers and the switches. To solve these issues, Virtual Networks (VNETs) are used instead. In this scenario, physical machines communicate with each other over Ethernet protocols, and virtual ma-

chines use VNET to encapsulate the packets. VNET technology guarantees that each packet is isolated from others VNET, being transparent to the virtual machines.

From the point of view of simplification of deployment, in a data-center physical machines resides in the same Layer 2 network and typical infrastructure has components dedicated to management, monitoring platforms, routing protocols, all of them making use of the network. Furthermore, the environment in the Cloud is highly flexible, large-scale, and each physical machine can accommodate tens of virtual machines. As such, a protocol such as Diverter [14] solves these issues using fully distributed routing system to achieve "one hop" communication between virtual machines. Even though this problem seems to be solved at the virtual machine layer, infrastructure components and services need to be aware of the Layer 2 problem. The authors of Diverter [14] also clearly states that if broadcasts are used, it is unmanageable to run the infrastructure above 1000 nodes, and even simple mechanisms of DNS resolving is a burden at a large scale. In conclusion, management services should be carefully designed, preventing broadcasts even for monitoring.

Another concern is related to the fault tolerance and bandwidth requirements of the routers within the network. Typical solutions rely on redundant hardware switches in a hierarchical division in which the top most switches are more capable than the lower ones. This division leads to expensive solutions because the top most switches are not commodity hardware.In [3], the problem is solved using only commodity parts, using a called "fat-tree" design, in which redundant links are created in order to maximize bandwidth.

2.3.3 Storage Virtualization

With respect to storage, cloud management services need to provide the virtual machines with a pool of block devices that are transparently used. These storage devices must have an high degree of availability, as most nodes depend on the storage devices to work. In a data-center enterprise infrastructure, is frequent the use of SANs and NASs to provide blocks of storage to the users, but this *per se* does not meet the requirements of fair resource sharing and isolation. Currently, management services have a thin layer of logic to provide Virtual Storage Devices (VSDs) to virtual machines that satisfy the guarantees stated above.

However, in most scenarios, namely in PaaS, other types of storage

are offered to the customer. As the virtual machines/applications are typical read-only data, higher level solutions exists to fulfill most of the storage requirements, such as databases, object stores and messaging queue systems.

Besides the topic off how to distribute the storage space to the virtual machines, there is another concern on data reallocation at a large-scale scenario. The instantiation of a virtual machine involves to bulk copy a previously configured bundle to the physical machine. This poses many problems in terms of bandwidth utilization. Solutions that involves caching and explore opportunistic placement begin to appear in the literature, such as SnowFlock [31].

2.3.4 Services

In addition to these services, there are other critical services in the internals of the Cloud that constitute problems due to the large-scale deployment. One of this services is the configuration management at each server in the Cloud. For this work, SmartFrog [22] is a framework that leverages the deployment of components in a distributed and clean fashion. It enables to deploy the desired services for the Cloud using orchestration, which means the life-cycle of components is automatically managed. This is critical to Cloud management services whose components rely on the deployment of others in order to be started.

Most Cloud providers also provide monitoring capabilities to the deployed resources, having the customers to possibility of watching themselves in (almost) real-time the state of the different resources. Such mechanisms already exist in most data-center deployment for internal use, whose reference implementations are Nagios and RRDTool. However, at this scale, other protocols emerge such as PRISM [26], Ganglia [33] and the PlanetLab monitoring tool CoMon [39].

2.4 Open-Source Project

2.4.1 Eucalyptus

Eucalyptus [37] is an open platform provider for IaaS, namely for use within private clouds. Eucalyptus was primarily build atop Xen [9], having no concerns on network isolation and implementing Amazon

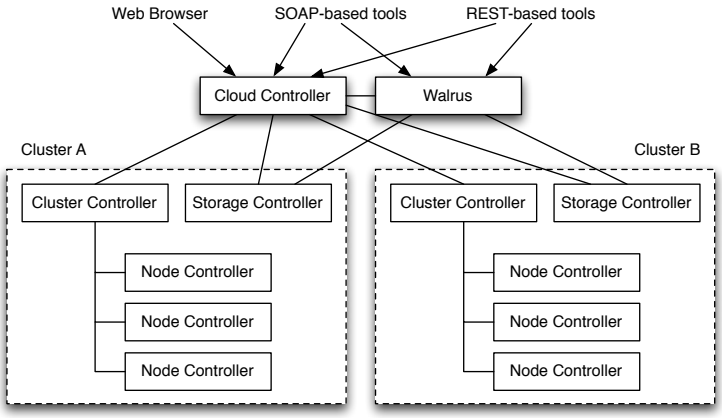

Figure 2.2: Eucalyptus Cloud architecture.

EC2 interface, namely the SOAP and Query interface based on Web Services. By implementing this interface, Eucalyptus soon became popular and a reference in the open-source world. To Eucalyptus was also given a great boost from the fact that it became the Cloud Computing reference for the Ubuntu Linux distribution.

Nowadays Eucalyptus is a mature IaaS platform, that provides an end-user solution for a Cloud Computing environment. Eucalyptus is composed of five major building blocks, each one with a concrete role.

Following Figure 2.2[1] from top to bottom, the first component is the Cloud Controller (CLC) that exports the services of the Cloud to the end-user. This is an entry point for all the requests to the Cloud, which can be done not only by Web Services as referenced above, but also from a Web interface developed in the Java programming language. All the requests are further redirected to one of the Cluster Controller (CC) which schedule the virtual machine instantiation to a free Node Controller (NC). While these components of the platform have a clear and well defined role, Eucalyptus has the flexibility of allowing more than one role at the same physical machine.

The other two key components are the ones responsible for the Storage devices in the Cloud. Walrus is a put and get storage service, similar to Amazon S3, while the Storage Controller provides the virtual machines with disk blocks, similar to Amazon EBS.

[1]Image released under Creative Commons Attribution-Share Alike 3.0 Unported license, authored by Stevenro

Chapter 3

Problem Statement

This Chapter enlights the reader about the problem focused on this dissertation. The first part carefully describes the target scenario that we are focusing on, namely detailing some of the open issues refered at Chapter 2. Then we analyze the requirements that a Cloud Computing management service should meet, clearly stating the rationale behind them. Finally, we compare and contrast some key Cloud providers and explain why they do not satisfy these requirements.

In a typical Cloud Computing infrastructure, we have a rack-mounted system, comprising thousands of equally powered nodes, making the Cloud a large scale infrastructure. Typically, internal rack communication is cheap, but global communication in the data-center needs to be carefully addressed. This derives from the fact that communication between different racks involves the use of core routers from the data-center and consequently more hops on the network. Sometimes, a Cloud infrastructure is spanned across multiple data-centers, and different logical locations, posing new problems on synchronization among different regions.

We assume that the network for the management service is not dedicated. The rationale behind this decision is that it would require separate hardware dedicated only to management, which will compromise the adoption of the solution due to its cost. Hence, the bandwidth usage is shared between regular network traffic and management one, so service should cause minimal impact. As stated in Section 2.3.2, broadcast on a large scale data-center is unmanageable and we do not assume the existence of multicast. Special network hardware with processing capabilities and high buffer, was not considered in this scenario, because such commitment would not

be profitable.

With respect to a typical load of a management service for the Cloud, the statistics are understandably not available to the public. The public naive study published on [42] and [49] is believed to be a good estimation of a workload on Amazon EC2 [5]. It analyzes the anatomy of the resource IDs, estimating the number of instantiations during some period. Following this study, at 2009, Amazon EC2 was receiving request of above 50,000 instances (creation of virtual machines) per day. These comprises only the resources provisioned on the Cloud, and naturally we expect read-only operations to be more frequently. This load is a base target for our scenario.

The first step on analyzing the requirements is to look at the perspective of the users of the Cloud. In case of a Cloud Computing platform we have to determine the key functional requirements on the perspective of the customer and from an administrator perspective.

From the point of view of the client, a cloud is an isolated environment in which he/she has the ability to instantiate resources, concretely virtual machines. This process proceeds with a client describing the components of the desired virtual machine in a XML file that the Cloud provider has to instantiate. Most of the providers facilitates even further this instantiation, offering the use of Web services, Web-based WYSIWYG and custom interfaces. In most of the interactions, users can bypass the configuration step and choose from pre-configured virtual machines for common roles (eg. databases or Web servers). Another key aspect is that the customer sees the management as a mean to define and control instances, and perceives that a failure on the service should only influence these aspects, not the Cloud as a whole. By this we mean that, in case of a disruption of the management service, virtual machines already deployed should remain running.

From the point of view of an administrator of the Cloud, a datacenter is a complex infrastructure, a system where change happens continuously. Changes can occur to due to many factors, some of them occur from a normal operation of the Cloud, while others occurs from unexpected factors. The normal factors comprise the ones that occur from the elasticity property of the Cloud: as the Cloud is able to add and remove resources transparently to the client, it must also be able to add and remove physical nodes, being able to up/down-scale as needed. Furthermore, there are some scheduled down-times, to perform system maintenance and reconfigurations

that should not disrupt the Cloud services. Other factors such as hardware failures and catastrophic failures also affect the Cloud in a unplanned mode. Occasional hardware failures should be dealt in a transparent manner, moving the resources instantiated; and catastrophic failures must retain information of what is already deployed and degrade gracefully.

Stating this vision, a Cloud management service must be able to deal with a large scale system in a constant change. By this we mean that to add dependability to a Cloud Computing platform requires scalability, resilience and availability. To fulfill this perception that we have on a Cloud, follows a summary of the requirements that a management service must take into account:

1. Maintain currently deployed resources in case of management service failure;
2. Provide re-instantiation of virtual machines on a node failures;
3. Provide an unified view of the system, both to data-center operations team and to the outside world;
4. Ability to add and remove physical nodes to the system, being able to up/down-scale when needed;
5. To be oblivious to hardware failures and be dependable and safe on catastrophic scenarios;
6. To be able to deal transparently with multiple location deployments;
7. Scale on the number of nodes and requests.

The work of this thesis is focused on the definition of an architecture for the management service in a Cloud Computing environment that meets all these requirements. We are certain that these requirements are of an extreme importance in two different perspectives. The former is to improve the adoption of the Cloud Computing paradigm in scenarios where dependability is an essential requirement. The later is to leverage the administration at Cloud data-centers to automated and autonomic systems.

This research topic is also considered prominent in the literature. The problem of dependability and scalability of the Cloud infrastructure is a key element of research as stated on the report of LADIS 2008 [10], whose topic was Cloud Computing. As a consequence, the

management service must follow the same principles of a dependable infrastructure to achieve the same level of confidence. The results of the conference give some insight on the challenges that we are facing, namely searching for the better architecture in order to provide the most adequate solution.

Despite these requirements naturally emerge from a detailed analysis of the Cloud Computing, these requirements are not assured by mainstream Cloud providers, which attest the relevance behind this work. The management service in contemporary Cloud providers is usually addressed on a best-effort basis, not guaranteeing all requirements the stated above. In Amazon EC2 [5], instances are not guaranteed to survive an hardware failure, as the customer is responsible for the management of each virtual machine. Other solutions as Eucalyptus [37] are merely based on hierarchy of management servers. This platform does not account for dependability, as the each management role on comprise one physical node. In this solution, a missing component such as the Cloud Controller compromises all the system, being a Single Point of Failure.

Chapter 4

Architectures for The Cloud

This Chapter introduces a set of architectures for the management service in a Cloud Computing environment. Firstly, we give an high-level overview of the conceptual model, which focus the reader on the overall system. Secondly, we propose different architectures and throughly examine them in terms of the requirements enumerated on the Chapter 3. And finally, we introduce some discussion on further possible architectures.

Before delving into details on different architectures, we give some intuition on the management system, combined with some key concepts to familiarize the reader.

4.1 The Management System

Figure 4.1 presents an abstract view of the management system of a Cloud Computing infrastructure. The system has a well defined external entry point where client requests arrive. This is called the Front End Layer, and is typically composed of Web/Application Servers that interact directly with customers. Later on, the request is submitted to the Management Layer, composed of one or more Management Servers, which are responsible for choosing the node where the virtual machines will be deployed. The Back End layer is a pool of servers, called Worker Nodes, available to serve customers requests.

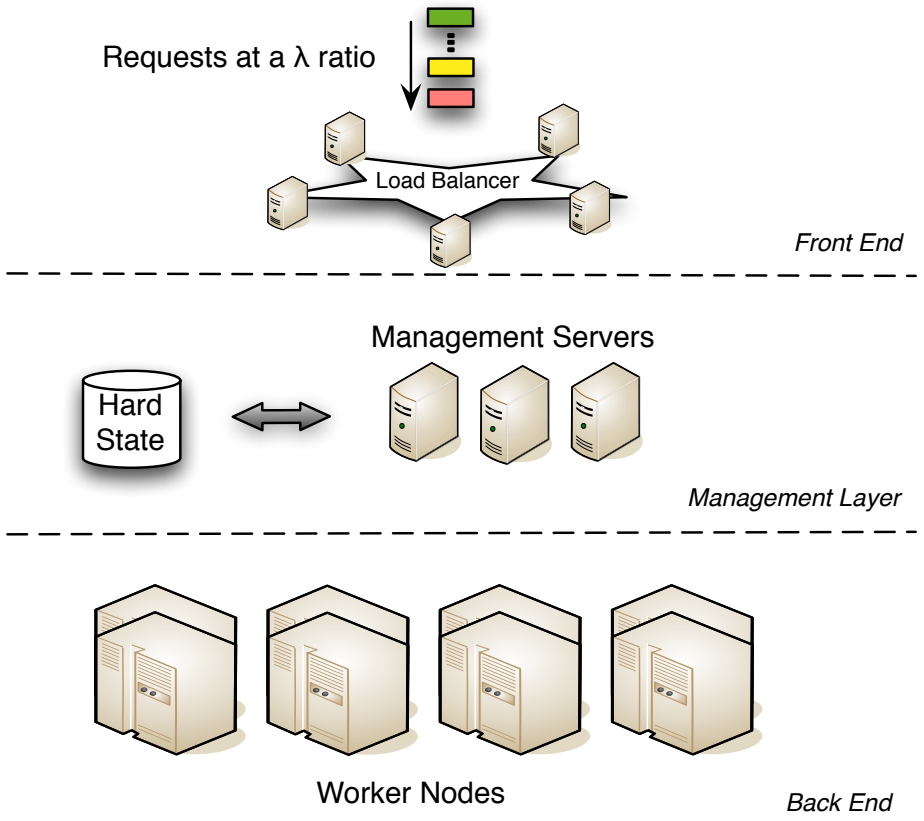

Figure 4.1: Cloud's management system overview.

4.1.1 Key Roles

Building an architecture for the management service involves choosing from the pool of available servers: 1) which ones are part of the management itself (Management Servers); 2) which are the servers that host customer virtual machines (Worker Nodes). The distinction between Management Servers and Worker Nodes is crucial to the understanding of the roles they play in the architecture.

On one hand, Management Servers are those who are responsible for taking care of other servers state. The main goals of these servers are: 1) instantiate the resources requested; 2) maintain the current state of the nodes they manage; 3) account for the presence or absence of managed nodes, and acting on failures accordingly. To maintain a cache of current state of each node, a Management Server relies on polling the node or waiting for state change to occur. Using this information, locating instances can be made at this layer. To be aware of node failures, the Management Server uses an heartbeat protocol to monitor Worker Nodes's availability. The nodes allocated

to the Management Layer should be kept at a minimum, maximizing the nodes producing meaningful work.

On the other hand, Worker Nodes, being more numerous, supply the computing power to accommodate all the customers resources. Specifically, Worker Nodes are virtualization hosts that are able to deploy virtual machines, using Xen [9], KVM [29], VirtualBox [38], VMware [48], among others virtualization platforms. A thin software layer is running at each Worker Node at the privileged domain to exchange heartbeats and control messages with the Management Servers, and to accomplish the tasks triggered by the Management Server, namely the provisioning / unprovisioning of virtual machines.

4.1.2 Soft State versus Hard State

At a Cloud Computing platform, providers must account for the instances already deployed. Ensuring that this information is durable, allows this information to be used in case of a failure and, mostly important, to be available on a catastrophic scenario.

In the management service there are two distinct states of the system, and realizing this difference is one of the keys to reach dependability. One comprises the state of current instances running - Soft State - despite failures of Worker Nodes. This state, kept at Management Servers, is extremely volatile as it matches the current running resources. On a Worker Node failure, instances that were hosted in it need to be reinstantiated, using for this purpose the Soft-State information. However, in certain situations simultaneous failures at Worker Nodes and Management Servers can occur. In that case, after the Management Server recover, it needs to check somewhere else this information. To checkpoint this information, the state of instances is kept separately on the Hard-State. This Hard-State store the virtual machines information in a consistent and durable fashion, being accomplished by a database management system, which guarantees these requirements. The Hard-State is not as volatile as the Soft-State, as this state is only changed upon creation or removal of instances using transactions and read upon Management Server failures.

4.1.3 Operations in the Cloud

Operations submitted to a Cloud management service mimic a Create, Read, Update and Delete (CRUD) interface, applied to virtual

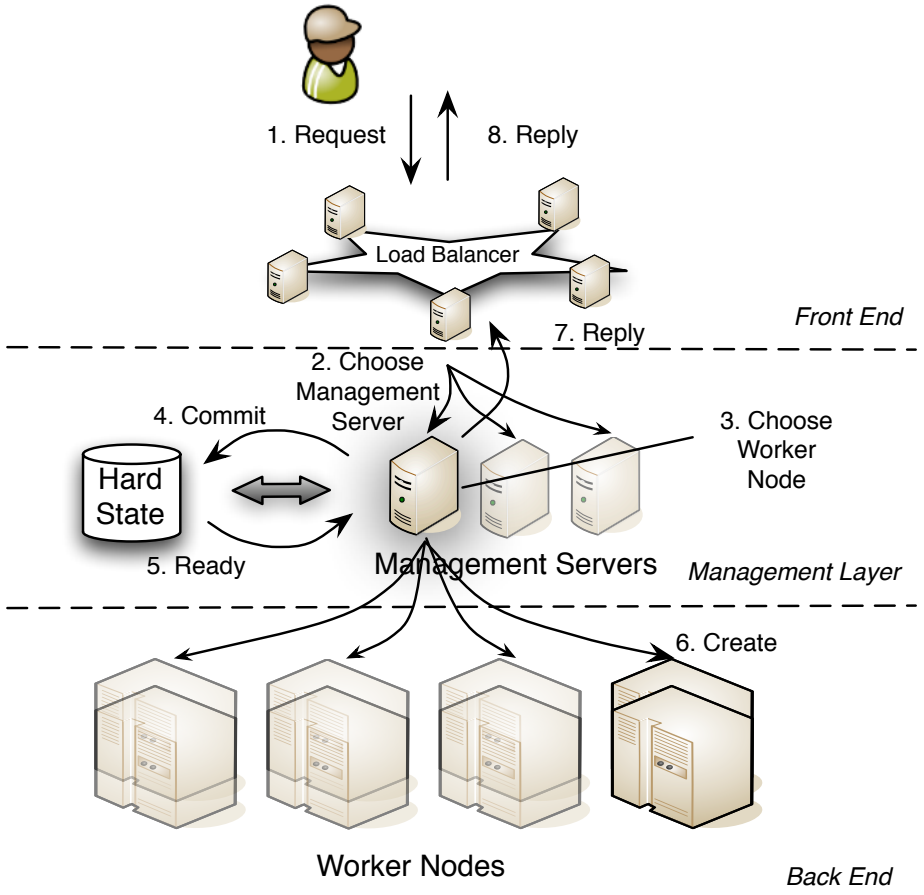

Figure 4.2: Algorithm of the Create operation.

machines. Typically, for each virtual machine configuration, several instances can be created, with the same configuration properties. So, an instance is a deployment of a virtual machine in the Cloud, being identified by the machine ID, that varies from Cloud to Cloud platform. An instance can be deployed - Create, be accounted for its state - Read, have its resources updated - Update and terminated - Remove.

In Figure 4.2 we present an illustration of the algorithm for the Create operation. This operation is the most elaborated operation, which involves using both of the Hard and Soft-State.

From Figure 4.2, we can identify these steps on the Create algorithm:

1. The customer requests the creation of a new instance of a virtual machine, contacting one of the Front End servers.

2. The Front End server chooses a Management Server available

to satisfy the request;

3. The Management Server looks up current state for resources, choosing a Worker Node to create the instance;
4. The Management Server informs the Hard-State that a new instance will be deployed at the chosen location;
5. The Hard-State replies to the Management Server;
6. The instantiation order is sent to the Worker Node;
7. The reply is sent back to the Front End;
8. The Front End server replies to the customer.

With respect to other operations, they follow a similar pattern and therefore we do not need to present them here.

4.2 Centralized Management

The first approach is to allocate only one Management Server to the management service, as shown in Figure 4.3. This seems to be a reasonable approach to begin with, as we are trying to reduce the resources dedicated to management.

In this centralized architecture all the requests are directed to the Management Server, which is a well-known entry point. Upon request, the Management Server has global knowledge of resources instantiated and is able to decide where to deploy the new instance. Later, it follows the algorithm described in Section 4.1.3 to create the virtual machine. In terms of configuration, this is reduced as at the client-side there is a single entry point, and all the Worker Nodes contact the same Management Server.

Using only one node to perform the management requires an additional effort in order to fulfill the requirements of Chapter 3. Focusing on node failures, two different scenarios can happen: the first is a failure of the Management Server and the second is failures of Worker Nodes. On the first case, the failure of the Management Server leads to a down-time in the management service, but Worker Nodes are ready to maintain already deployed resources. When this happens, the Management Server needs to be manually replaced and synchronize its state before it becomes available: it needs to reconstruct the state of running resources and then check with the Hard-State

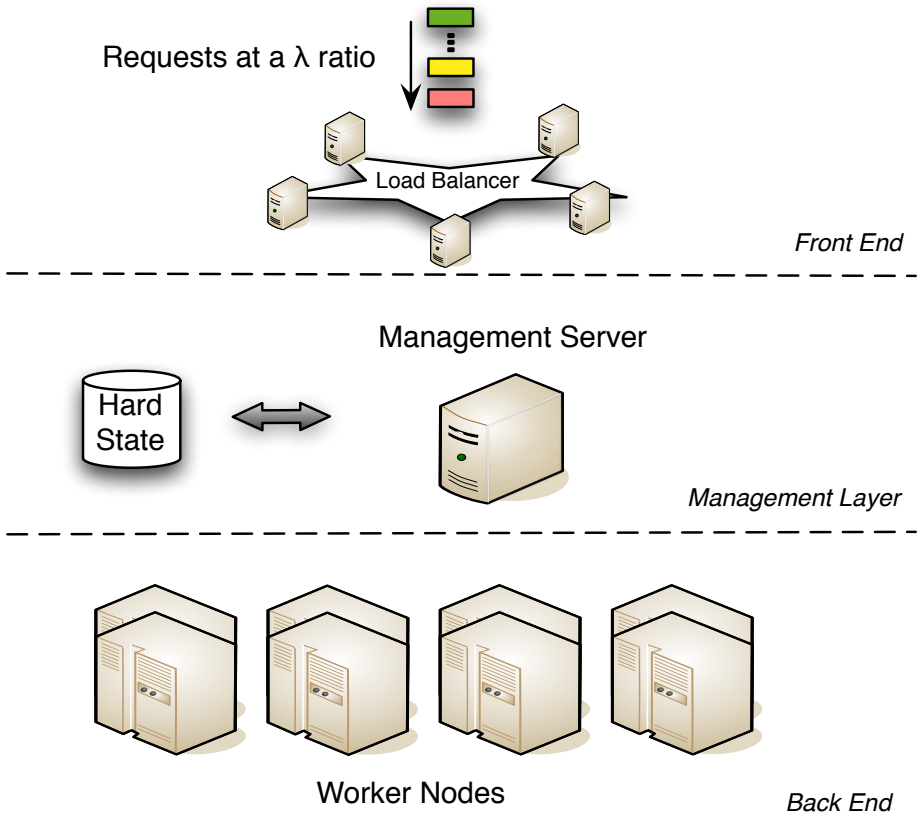

Figure 4.3: Centralized architecture.

to make sure everything is properly deployed. On the second case, for each Worker Node failure, the Management Server needs to redeploy the resources in other available Worker Nodes, updating this information at the Hard-State.

The most advantageous aspect of a centralized approach is that a single Management Server offers an unified view of the current system. The Soft-State included at the Management Server is sufficient to provide information such as currently running instances and availability of Worker Nodes.

An important property of a management service is also its safety. On a centralized architecture, the Management Server is fully confident on its state. This clearly implies that safety can be assured if there is only one Management Server at a time. In a catastrophic scenario, if the Management Server is faulty it must be replaced manually to prevent multiple Management Server, as they are not ready to run concurrently. In terms of dependability, the resilience of the management service is limited to the Management Server node, which is

clearly insufficient. While we do not rely on special hardware, in a real deployment the Management Server reliability can be improved using redundant hardware. Still, it is a single point of failure.

The centralized solution is not scalable by design. A single Management Server is not able to scale on the number of nodes it manages and requests. While it is not clear the upper bound on the scalability of this architecture, processing power and network grows linearly with the number of Worker Nodes and requests. One one hand, in a scenario with a large number of nodes, the Management Server needs to account for more Soft-State and to receive more heartbeats to keep the state updated. On the other hand, a large number of requests saturates the Management Server as for each request a connection needs to be maintained, and the resources allocated to fulfill the request. As in the previous paragraph, if we admit special hardware to the Management Server, the problem is only postponed until the Management Server reaches its limit.

To sum up, a centralized approach is balanced in terms of the requirements it fulfills. Maintaining deployed resources (requirement 1), providing re-instantiation on failures (requirement 2), providing an unified view (requirement 3), being able to add and remove nodes (requirement 4) and accounting for safety on catastrophic scenarios are successfully addressed by a centralized solution. However, the architecture fails on scalability (requirement 7), not providing a solution to a ever growing number of nodes and requests at a Cloud Computing environment. The dependability and resilience of the management service (requirement 5) is compromised, as the Management Server is a single point of failure. The requirement of multiple location deployments (requirement 6) is not considered in this analysis.

4.3 Decentralized Management

In order to meet dependability and resilience at the management layer, a distributed approach was considered. In this solution, all nodes represent both of the roles of Management Servers and Worker Nodes, as shown in Figure 4.4.

The decentralized architecture provides a solution in which all the nodes are organized in a peer-to-peer fashion. By this we mean that nodes have equal roles in the system, having the same share of responsibility on the management. All this group share the Soft-State, which means this group has replicated information on cur-

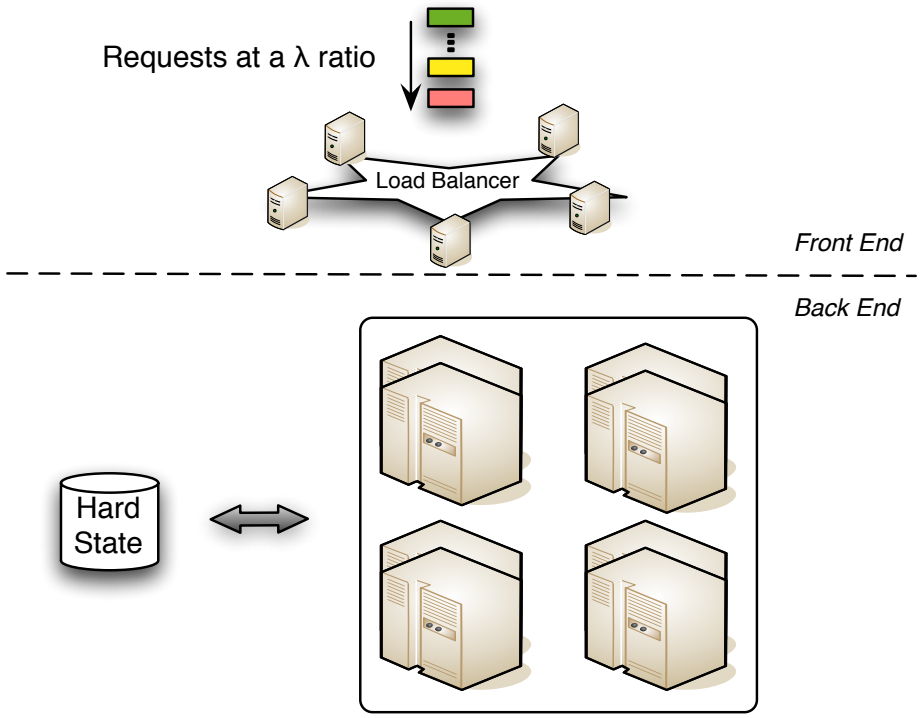

Figure 4.4: Decentralized architecture.

rently instantiated resources. To implement this replication process, the literature provides several alternatives, namely active [44] and passive [11] replication. The passive replication is adequate in this scenario because operations cause a side-effect (commiting to hard-state), thus they are done on one replica and then propagated to other nodes. The design choices here are very subtle, but all of them imply applying transactions to the Hard-State and then synchronously propagating these changes to the Soft-State. Retaining this information is sufficient to pinpoint some problems on scalability later on.

This solution involves the use of a Group Communication Service (GCS), such as JGroups [8] or the Spread Toolkit [6], to provide state replication. The use of a GCS, facilitates the elasticity of the Cloud, as the nodes can be dynamically added or removed to the group. It also reliefs the implementation from Group Communication details, such as electing a leader and node churn management.

In this architecture we cannot evaluate the management server reliability alone, but the Cloud system as whole. Thus, as long as there is a non-faulty node in the system, the management service is ready to receive request. Every time a node fails, the coordinator chooses

from the group of nodes the place to redeploy the resources, proceeding accordingly. Using a coordinator prevents the other nodes from redeploying the resources, which would lead to an inconsistent state of the system.

The decentralized approach provides an unified view of the system at each node. As each node contains the Soft-State replicated from the coordinator, any node can be considered to consult the current state of the Cloud Computing environment. By this we also mean that read operations can be redirected to any node, as the knowledge they have on the system is consistent and global.

The use of a GCS is advantageous because it brings consistency to the state shared by the nodes, but also has some drawbacks. In case of failures of a single node failure, this architecture behaves as expected, removing the node and redeploying the resources as necessary. However, in case of a catastrophic scenario, partitions can occur. Following the CAP Theorem [20], we cannot achieve all the three guarantees of Consistency, Availability and Partition Tolerance. Bearing this in mind, in case of partition the system should stop, preventing each partition to continue to run independently. This would lead to a redeployment of all the resources in each partition, which is not acceptable. To sum up, in case of a partition, the management system stops until a majority of nodes is achieved again.

In a decentralized approach we can start to envision a scenario with multiple location deployments. A simple migration of this architecture to multiple location deployments in which all nodes form a group is not feasible as it would require enough bandwidth between different locations, incurring the system with prohibitive costs.

In what refers to scalability, this architecture does not solve the problems of the centralized approach, being even worse. An increment on the number of nodes on the system implies joining it to the GCS and consequently raise the number of control messages needed to maintain the group state consistent. This incurs in a penalty at each node and also increases the network usage if a typical point-to-point protocol is used. Achieving scalability on the numbers of requests is also difficult in this architecture, namely the write ones. For instance, for Create operation the coordinator needs to replicate the information among all nodes.

Summarizing all the stated above, the decentralized solution adds dependability to the management service, but still fail in terms of scalability (requirement 7). With all the nodes making part of the man-

agement service we do not need do deal with management service failure (requirement 1) because in that case everything has failed. The reinstantiation of virtual machines (requirement 2) is addressed by the coordinator elected from the GCS, which transparently manages the addition and removal of nodes (requirement 4). The unified view of the system (requirement 4) can be obtained at any node as the state is synchronized. On hardware failures (requirement 5), the system behaves as expected, but catastrophic scenarios can lead to partitions, halting the management service. While this architecture could be deployed on multiple locations (requirement 6), that would not be feasible due to high network usage.

4.4 Fixed-size Management Group

In this solution, we try to get the best of both worlds, having an intermediate solution. We present an architecture in which there is a fixed-size group of Management Servers, as Figure 4.5 illustrates.

In this architecture, we reduce the impact of having all the nodes managing the system, reaching a fixed-size group of Management Servers. Finding the right number of the group is not a trivial task since this differs from number of overall nodes. One can state that a small percentage of nodes (say 10 per cent) can be allocated to the management service; however, in a pool of one hundred nodes this number might make sense, but does not in a pool of millions of nodes. The management service works in the same manner of the previous approach, but in this situation there are Worker Nodes being managed by a small number Management Servers.

To be able to accomplish this approach, special care needs to be taken into account by the operations team. Thus, being the Management Servers fixed-size, they need to be replaced on hardware failures to maintain the management service running.

In this approach, the service become inoperable when all the Management Servers Servers fail. In that case, already deployed resources remain instantiated until a new Management Server arrives. On bootstrap, the Management Server verifies if it is the only one on the system: if it is, it checkpoints the instantiated resources with the Hard-State; if it is not, it gets the state from one of the running Management Server. In case of a Worker Node failure, the coordinator has the responsibility of redeploying the resources in other Worker Nodes.

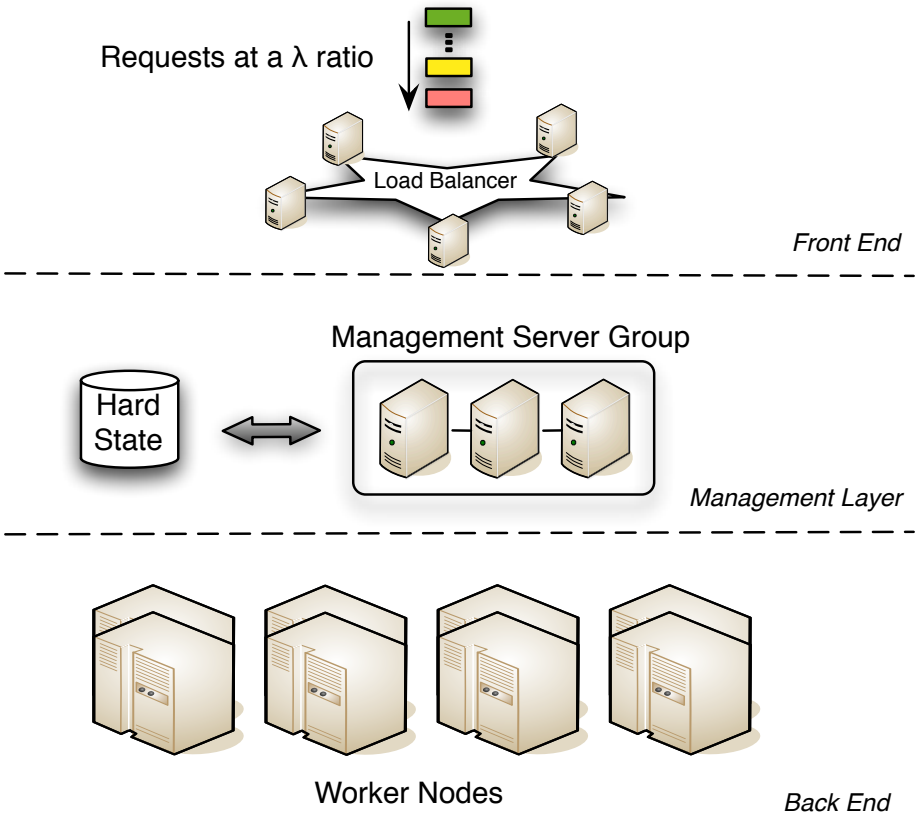

Figure 4.5: Fixed-size group architecture.

The unified view of the system is provided at each Management Server, as they have global knowledge similar to that of the decentralized architecture.

The addition and removal of Worker Nodes to the system is handled through updating the Soft-State at the Management Servers. These life cycles update only the Soft-State of the system as they do not interfere with the GCS.

The dependability of this solution is confined to the nodes that constitutes the management layer. The reliability and availability of the system tends to be lower than the previous solution, as less nodes are involved in management. From one point of view, we could distribute the Management Server along the data-center in different racks, which raises the independence of failures, diminishing the probability of service disruption. From another point of view, Management Servers can be placed at the same rack, within an easier access location, and be clearly identified to the operations team as critical to the system. This solution also suffers from partition prob-

lems, specially in the former case.

This solution still requires enough bandwidth across datacenters. When compared with the previous scenario, these requirements are reduced, due to a lower number of Management Servers.

In terms of scalability of the system, this approach is better than the previous one, as there are less nodes involved in management. This means that a coordinator need to contact a lesser number of nodes to fulfill a request. As in the centralized architecture, Management Servers can be deployed in dedicated hardware to improve the scalability. In spite of being a more scalable solution, this solution still implies that the Soft-State is shared, and synchronization is needed.

In conclusion, this solution involves fewer nodes on the management layer, without compromising the dependability of the service (requirement 5). In light of the requirements stated at Chapter 3, 1, 2, 3 and 4 are satisfied in similar terms to the decentralized approach. This architecture brings improvements on multiple locations (requirement 6), while not being perfect. The scalability (requirement 7), while better than decentralized approach, still relies on synchronization, so it is limited.

4.5 Space-split Management Group

In this approach, we dedicated our effort in making the architecture scalable. For this purpose, we aggregate the Worker Nodes in several containers which we give the name of buckets, each one having a Management Servers, as it is shown in Figure 4.6

This solution also comprises a fixed number of Management Servers, but organized in a different way. We divide the nodes by buckets, each one having a well defined Management Server. This distinction can match the possible infrastructure of a data-center, using all the nodes at a rack to form the bucket, or use a more dynamic mapping, using several well-known hashing algorithms [30]. We can see the solution proposed in Section 4.2 as a building block of this solution, as each bucket comprises one Management Server and a number of Worker Nodes. Moreover, all Management Servers still constitutes a group, but in this case the Group Communication Service is only used to maintain the group membership, because each Management Server has a different set of Worker Node to manage. The group membership is needed in case of failures a Management Server, that will evolve the replacement of the Management Server with other

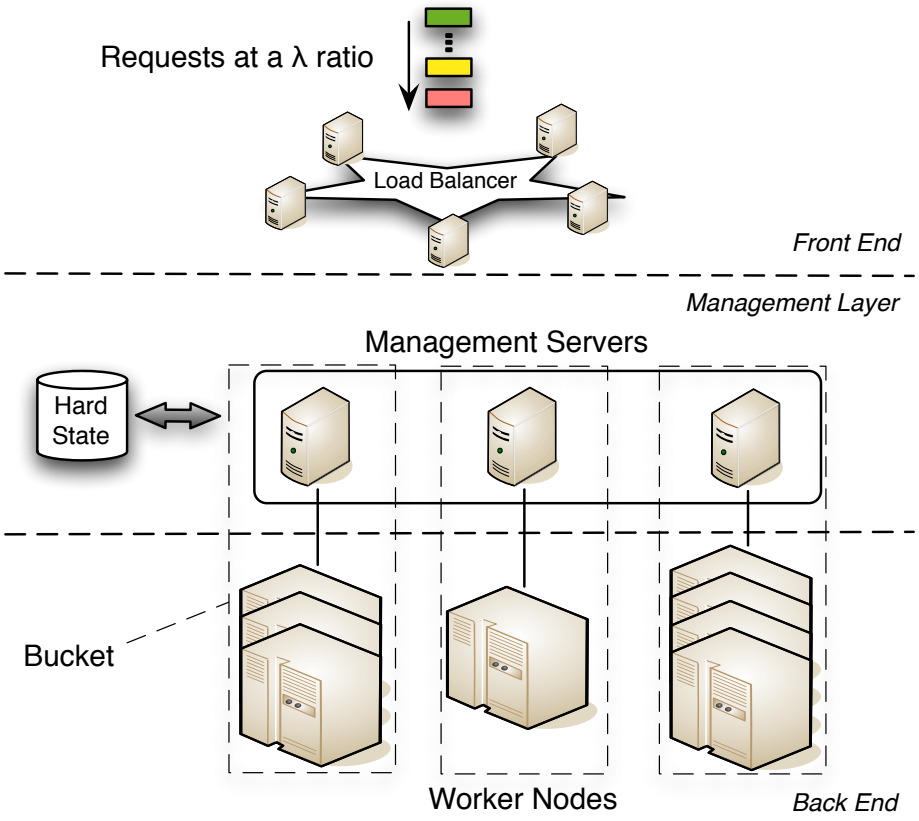

Figure 4.6: Space-split group architecture.

available. The improvement of this architecture derives from the fact that Management Servers only need to exchange control messages on membership changes, not in all the requests, as the previous decentralized approaches.

This space-split architecture avoids maintaining the Soft-State synchronized, as in a decentralized approach. This implies that on a Management Server failure, the system cannot automatically rebalance the Worker Nodes, but needs to recover from the Hard-State, as only the Hard-State has all the information consolidated. In case of a Management Server failure, the system cannot instantiate new resources on the respective bucket, but the management service remains operational. On all Management Server failures, the resources are maintained deployed until the Management Servers are restored.

Following the same logic of the centralized approach, each Management Server is able to re-instantiate virtual machines on Worker Node failures. This re-instantiation can be assured by a Management Server until its bucket has enough resources; otherwise the

Management Server needs to contact another Management Server to fulfill the request.

The price to pay for having this architecture is that we lose the unified view of the system, as each Management Server only views a portion of the Cloud infrastructure. While this a clear drawback, it might not convey a real problem in large-scale deployments, in which multiple operations teams exist, and each team manages an independent bucket.

Addition and removal of physical nodes involves nothing more than the previous approaches. An initial configuration setup is required to bootstrap the Worker Node to join it to the correct Management Server. To add and remove a Management Server from the management layer we rely on the Group Communication Service.

This solution still gives the same guarantees as the previous solutions in terms of dependability. As stated earlier on this section, the Management Servers form a group in which there is not a shared state, each one having a slice of the Soft-State. The Management Servers account for failures and rebalancing the Worker Nodes as needed. In case of failure, the Soft-State can be reconstructed from the current state of the Worker Nodes, and the Hard-State can be reconstructed from the database. It should also be pointed out that a failure of a Management Server only provokes the disruption of that manager, and it only remains until the failure is detected, and the others Management Servers take over control.

As the group of Management Servers only maintains group membership, multiple location deployment can be achieved, as the system does not require a lot of bandwidth. This improves the previous solution that requires synchronization not only on group membership, but also on instance provisioning.

In terms of scalability, this solution improves the previous architectures, as the management is splited through several Management Servers, being more network-friendly. Instantiations can be decided locally and do not need propagation to other Management Servers, which is an improvement in relation to decentralized approaches. For the first time in these architectures we are not limited to the throughput of a single server or coordinator, so this solution scales in number of Worker Nodes and on requests, as long as the group membership scales.

Retaining all this information, requirements 1, 2 and 4 are addressed by this architecture, as explained above. The unified view of the system (requirement 3) can pose a problem in this architecture, while

not risking the usefulness of this approach. In relation to multiple location deployment (requirement 6), this comprises the better approach of all architectures analyzed before. The dependability (requirement 5) and scalability (requirement 7) is fulfilled, surpassing the previous architectures.

4.6 Hierarchical Management Group

While the previous solutions are reasonable for our scenario of thousands of nodes at a data-center, this might not be a solution with millions of nodes. Analyzing the previous solution, this solution can grow in scale as long as we can maintain the group membership of Management Servers. This only imposes a limit at a very large scenario, that is not find on todays data-center. Thus, we can devote some time to find a conceptual solution for this order of scale.

Being inspired by Astrolabe [46], we envision an architecture that comprises an hierarchical approach, while having dependability in mind. Buckets from the previous Section can be seen as the leaves of a leaf-tree. When two or more buckets are joined, they elect a coordinator to create an abstract group, which comprises the parent layer. This is recursively from the leaves up to the root.

An example of the proposed architecture is depicted in Figure 4.7. We have four Management Servers A, B , C and D, representative of an arbitrary number of Worker Nodes, which forms the nodes. At each level of the tree, a coordinator is elected as the parent node, proceeding in the same manner until the root of the tree. Management Server B has a full information relative of its Worker Nodes and Worker Nodes at Management Server A, and only an aggregated view of all resources that Management Server C represents. Management Server C has a full knowledge of its Worker Nodes and Worker Nodes at Management Server D, and an aggregated view of all children of Management Server B. Management Server A and Management Server D have information of its Worker Nodes and Worker Node of Management Server B and Management Server C, respectively.

Using this approach we achieve scalability through hierarchy (requirement 7), keeping the system dependable (requirement 5). The root of the tree does not provide a clear unified view of the system (requirement 3). However, it permits multiple location deployment in which the tree in distributed through different locations (requirement 6). Accounting other requirements, namely 1, 2 and 4 are

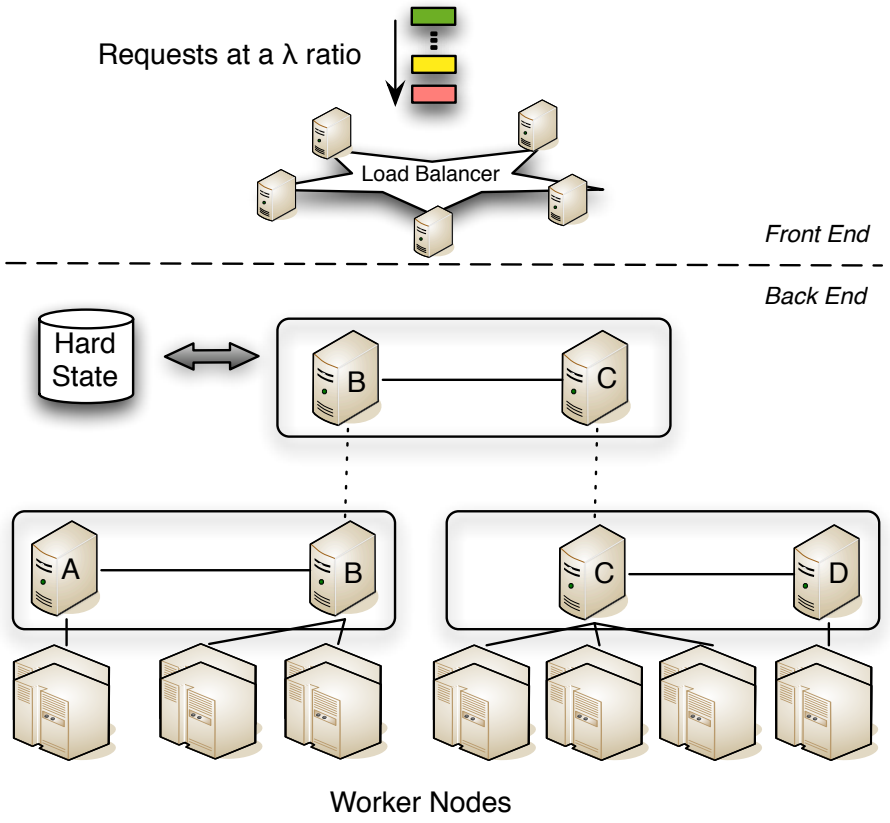

Figure 4.7: Hierarchical architecture.

addressed in the same manner as previous architectures.

Chapter 5

Experimental Evaluation

This chapter has three main sections. The first Section presents the different options that were considered in this evaluation, stating the rationale behind the tests. Section 5.2 introduces the experimental setup conceived to analyze the impact of the architectures. Finally, Section 5.3 presents and discusses the results from the evaluation.

Having in mind the requirements discussed on Chapter 3, on this Chapter we are attesting the usefulness of the architectures proposed in Chapter 4. While some requirements are assured only by reflecting them on the architecture design, while others need to be confirmed evaluating an experimental scenario. As a consequence, this experimental evaluation focus on the scalability on the architectures proposed.

5.1 Approach

Evaluating large-scale algorithms is always a difficult task. The evaluation of realistic environments is very expensive, time-consuming and difficult to reproduce. So, the natural approach to prove the usefulness of these architectures is to conduct a simulation, using the several toolkits available or build a customized simulator.

At a first sight, a customized simulator was seen as the right way to evaluate these architectures. To this intent, we use a custom made simulator using the Python programming language [40], which was built in-house, that allows rapid prototyping. This simulator allowed to quickly setup the experimental scenario, facilitating its manipulation . However, it was soon realized that this simulator only gives us insight on the correctness of the solution, rather than

pointing out performance issues. Consequently, we were not able to evaluate the architectures in terms of scalability, which is the most prominent aspect we want to evaluate.

For the reason stated above, we have to consider a real scenario, to measure the impact of the architectures in terms of scalability. To this intent, we studied available testbeds such as PlanetLab [15], that provides us with an inexpensive scenario and a large-scale environment. Nevertheless, PlanetLab does not account for the scenario we were looking for, as in PlanetLab the conditions are very different from the ones available in a typical data-center deployment: the resources provided in PlanetLab are shared among other tests; other simultaneous experiments causes interference with the tests; the network between nodes is WAN based, not reflecting a typical LAN network between Management Servers and Worker Nodes; and the nodes provided by the PlanetLab are not sufficient in number to the scenario proposed. With all these restrictions, the derived solution was to build an a real prototype for the Management Server and simulate the customers requests and Worker Nodes. With this prototype, we proceed to the evaluation of the scalability of the Management Layer, as described in the next Section.

5.2 Experimental Setup

The system under test is an implementation of the centralized architecture presented in Section 4.2. The implementation has three different modules: the Server, the Worker and the Client.

All the implementations were done using the Java [25] programming language. Java is a object-oriented language with a great adoption on the enterprise market. It has built-in primitives for concurrent programming, being a natural choice to achieve a scalable implementation. For these tests, we require a database, that constitutes the Hard-State. We used the MySQL [36] relational Database Management System (DBMS), with the InnoDB storage engine, which provides transactional guarantees. The MySQL stands for an adequate DBMS as it is widely tested and used in IT community.

5.2.1 Server Module

The Server module is a complete implementation of the Management Server, using the multithreading paradigm.

At each operation, the Server consults its internal Soft-State and, if needed, proceeds with a transaction on the Hard-State. To provide a transparent layer between Java and the database, the Java Persistence API (JPA) [51] framework was used. The implementation of choice was the Apache OpenJPA [18], using the Apache DBCP [17] for connection pooling.

The Server module also accounts for the state of the different Worker Nodes it manages, exchanging heartbeats with them. This was implemented using simple UDP messages between the Server module and the Worker module. The heartbeat algorithm is the most trivial one: the Server maintains a list of Worker Nodes that are currently running and the correspondent last time they have been seen, detecting failures on a certain pre-defined timeout. As a consequence, a message lost or duplication does not compromise the algorithm.

5.2.2 Worker Module

The Worker module comprises an arbitrary number of Worker Nodes, that contact the Server announcing its resources.

This module is implemented on top of the libvirt API [32], which provides an abstraction to interact with different hypervisors. However, in this scenario, we were not focusing on the performance on provisioning, so we used the libvirt fake hypervisor driver.

5.2.3 Client Module

The Client module is a simple class that stresses the Server with operations, simulating multiple customers.

This module can be configured with arbitrary time length between operations. For each operation, the module creates a thread that contacts the Management Server to perform the operation.

5.2.4 Hardware

For this experimental setup, two hosts were used, each one as HP Proliant with two AMD Opteron Processor 250 processors with 4GB of RAM, connected in a LAN with 1Gbps of bandwidth, running a Linux 2.6 kernel.

On one host, we deployed the Worker and the Client module, and the other host, the Server module is running, coupled with the DBMS.

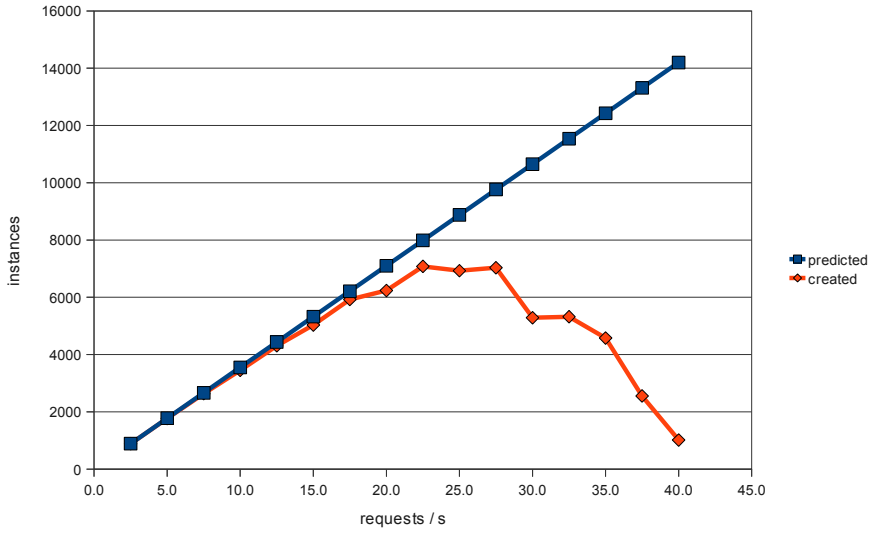

Figure 5.1: Number of instances deployed on the Management Server, at different request rates.

To be able to deal with a lot of active connections at each host, the Linux kernel was tuned to accept an unlimited number of open files and connections.

5.3 Management Service Evaluation

To evaluate the scalability of the centralized architecture, we conducted an experiment using 1000 Worker Nodes, with heartbeat announcements every 30 seconds. The Client used different request rates, ranging from 2 to 40 requests per second. The tests were done in runs of 360 seconds, cutting the data from the first and to the last 5 seconds, this ignoring the warm-up and cool-down periods.

The Figure 5.1 show the number of instances deployed at different request rates.

Analyzing the chart, we can observe that until the 20-25 requests per second, the system follows the predicted behavior, instantiating the same number of instances. Starting at that point, the number of instantiations does not go along with the predicted, having a declining behavior. Analyzing these results, some aspects can be taken into discussion. The first comprises the fact that at higher rates the system cannot scale on the number of request, starting to decline in the number of successful operations. The second is

that we predict that when the system achieves an upper bound of instantiations, it will proceed at that rate. However, the system behaves worse due to the fact that multiple customer operation poses to much stress on the Management Server and it cannot do its job. The third conclusion is that the Management Server does not have a predictable behavior during stress, which means that, in a system near the scale limit, the risk of failure of the Management Layer is high.

Thus, we can easily conclude that the centralized approach is not scalable on the number of requests.

Bearing this in mind, the decentralized architecture described in Section 4.3 can be shown to have the same or worse scalability than the previous architecture.

In the decentralized architecture, each node is part of a GCS. On create operations as the ones evaluated above, the operations must be executed by the coordinator and later on, propagated to the other nodes. Consequently, the scalability of this architecture is limited to the coordinator, which comprises a single server as in the above evaluation. Furthermore, the coordinator needs to wait for the shared state (Soft-State) to be synchronized among the nodes, so the performance of the system is limited by the slowest node on the infrastructure. In addition to these, in the decentralized architecture, each node play both roles of Management Server and Worker Node, meaning that each node is not dedicated to management, but also needs enough resources to provision customer virtual machines.

In relation to the fixed-sized management group architecture, described in Section 4.4, the same reasoning as the decentralized architecture can be done.

The fixed-size management group architecture has a group of Management Servers organized in a similar way of decentralized architecture. Though, the scalability is also limited to the slowest node in the Management Layer. This solution is better than the previous one, as the Management Servers are dedicated, so more load can be imposed on these servers.

The space-split architecture comprises a scenario in which each bucket is a single setup of the centralized architecture. So, this architecture constitutes a scalable solution.

Each Management Server at this architecture has its scalability limited to the number of requests stated above. However, a discretionary number of buckets can be added to the system, scaling out the solution. The Management Servers only share information on

group membership, acting only upon failures, which means this architecture can be scaled until the group membership is manageable.

Chapter 6

Conclusion

In this work we start by analyzing the requirements that a management service should have in order to provide reliability, scalability and resilience to a Cloud Computing environment. In order to promote the adoption of the Cloud, we carefully accounted for different aspects on the management service, namely replication, synchronization and coordination mechanisms, modeling the problem of a dependable Cloud Computing management service.

Later on, we propose architectures to a dependable management service. Using an iterative approach, we begin with a typical centralized approach, in which we have stated why that architecture did not provide a dependable solution. Then, we envision a totally decentralized architecture, which had failed on meeting scalability targets. Next, we describe a management service that comprised a fixed-size group that, while being more friendly on network requirements, still prevents the service to scale. The space-split approach, was the solution to a scalable service, in which we obtained some dependability. Finally, we present a conceptual view of an highly scalable solution, using an hierarchical approach.

Evaluation conducted at Chapter 5 proves our firsts suspicions towards the scalability of centralized architecture. Analyzing the results of our prototype implementation, we conclude that in order to reach scalability in a management service, we need to partition the management space between what we have called buckets. With this architecture, we have achieved a solution that is scalable by design.

To summarize, the main contributions of this dissertation were to provide architectures for the management service that avoid a single point of failure, such as the centralized solutions. Then, building a prototype, we were able to evaluate the rationale of these architec-

tures.

6.1 Future Work

The research conducted in this thesis poses some other challenges and open research towards new directions.

The architectures proposed, namely in Section 4.5 and 4.6, accomplishes great improvement in terms of scalability, providing solutions in which management space is partitioned. This solution requires group membership at the management layer, but its cost is reduced due to fact that the number of managers is small. If the management layer group goes beyond a certain scale, it will be reasonable to search for new approaches. We envision some group membership service that is scalable: instead of all managers having a complete group membership knowledge, each one should have only a partial view of the participants. As such, each manager will only account for the state of a bounded number of managers. Upon failure, an agreement is run between all members that were monitoring the failed manager to decide which one will be responsible for re-provisioning of the resources.

In the hierarchical approach presented in Section 4.6, the upper layers only needs an abstract view of the leaves to decide where to provision the virtual machines. This means that a request walks through the tree finding the best place to satisfy the request. A more interesting approach is to have some knowledge on the existent resources, and make more accurate decisions, such as provisioning virtual machines of the same customer nearby. Having an unified view of the system, it is relatively easy to apply these kind of policies. However, when there is no global knowledge on the Cloud infrastructure, this poses some new questions on how this can done. We foresee that by running some aggregation algorithms to exchange deployment information, we can achieve a scalable way of propagating the necessary information to the management layer, so that it can take more informed decisions.

Bibliography

[1] IEEE Standard 802.1Q-2003 Virtual Bridge Local Area Networks, May 2006.

[2] DC2MS: Dependable Cloud Computing Management Services. http://gsd.di.uminho.pt/projects/projects/DC2MS, 2008-2010.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, 2008.

[4] Amazon Web Services. AWS Case Study: Washington Post. http://aws.amazon.com/solutions/case-studies/washington-post/, 2008.

[5] Amazon.com, Inc. Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2, 2009.

[6] Y. Amir and J. Stanton. The Spread Wide Area Group Communication System. Technical report, The Center for Networking and Distributed Systems, May 1998.

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[8] B. Ban. JGroups: A Toolkit for Reliable Multicast Communication. http://www.jgroups.org, 2007.

[9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[10] K. Birman, G. Chockler, and R. van Renesse. Toward a cloud computing research agenda. *SIGACT News*, 40(2):68–80, 2009.

[11] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. *The primary-backup approach.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993.

[12] R. Buyya, C. S. Yeo, and S. Venugopal. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In *HPCC '08: Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, pages 5–13, Washington, DC, USA, 2008. IEEE Computer Society.

[13] K. A. Delic and M. A. Walker. Emergence of the Academic Computing Clouds. *Ubiquity*, 2008(August):1–1.

[14] A. Edwards, A. Fischer, and A. Lain. Diverter: a new approach to networking within virtualized infrastructures. In *WREN '09: Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 103–110, New York, NY, USA, 2009. ACM.

[15] P. Europe. Planetlab. http://www.planet-lab.eu, 2010.

[16] Flexiant. FlexiScale Public Cloud. http://www.flexiant.com/products/flexiscale, 2010.

[17] T. A. S. Foundation. Commons DBCP Component. http://commons.apache.org/dbcp/, 2010.

[18] T. A. S. Foundation. OpenJPA. http://openjpa.apache.org, 2010.

[19] J. Geelan. Twenty-one experts define cloud computing. http://virtualization.sys-con.com/node/612375, Aug 2008.

[20] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[21] GoGrid. Cloud Hosting. http://www.gogrid.com, 2010.

[22] P. Goldsack, J. Guijarro, S. Loughran, A. Coles, A. Farrell, A. Lain, P. Murray, and P. Toft. The SmartFrog configuration management framework. *SIGOPS Oper. Syst. Rev.*, 43(1):16–25, 2009.

[23] Google. App Engine. http://code.google.com/appengine, 2010.

[24] Google. Google Docs. http://docs.google.com/, 2010.

[25] J. Goslin and S. Microsystems. Java programming language. http://www.java.com, 1995-2010.

[26] N. Jain, D. Kit, P. Mahajan, P. Yalag, M. Dahlin, and Y. Zhang. PRISM: PRecision-Integrated Scalable Monitoring. Technical report, Department of Computer Sciences, University of Texas at Austin, May 2006.

[27] Joyent. Cloud Hosting Service. http://www.joyent.com, 2010.

[28] M. Kallahalla, M. Uysal, R. Swaminathan, D. E. Lowell, M. Wray, T. Christian, N. Edwards, C. I. Dalton, and F. Gittler. SoftUDC: A Software-Based Data Center for Utility Computing. *Computer*, 37(11):38–46, 2004.

[29] A. Kivity. kvm: the Linux virtual machine monitor. In *OLS '07: The 2007 Ottawa Linux Symposium*, pages 225–230, July 2007.

[30] D. Knuth. *The Art of Computer Programming, volume 3, Sorting and Searching*, pages 506–542. Addison-Wesley, June 1973.

[31] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Snowflock: rapid virtual machine cloning for cloud computing. In *EuroSys '09: Proceedings of the 4th ACM European conference on Computer systems*, pages 1–12, New York, NY, USA, 2009. ACM.

[32] libvirt.org. libvirt. http://www.libvirt.org, 2010.

[33] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30(7), July 2004.

[34] Microsoft Corporation. Azure services platform. http://www.microsoft.com/azure/default.mspx, 2010.

[35] P. Murray. Enterprise grade cloud computing. In *WDDM '09: Proceedings of the Third Workshop on Dependable Distributed Data Management*, pages 1–1, New York, NY, USA, 2009. ACM.

[36] MySQL AB. MySQL. http://www.mysql.com, 2010.

[37] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. pages 124–131, Washington, DC, USA. IEEE Computer Society.

[38] Oracle. VM Virtual Box. http://www.virtualbox.org/, 2010.

[39] K. Park and V. S. Pai. CoMon: a mostly-scalable monitoring system for PlanetLab. *SIGOPS Oper. Syst. Rev.*, 40(1):65–74, 2006.

[40] Python Software Foundation. Python programming language. http://python.org, 1990-2010.

[41] Rackspace. Rackspace cloud. http://www.rackspacecloud.com, 2010.

[42] G. Rosen. Anatomy of an Amazon EC2 Resource ID, Sep 2009.

[43] salesforce.com. Force.com. http://www.salesforce.com/platform, 2010.

[44] F. B. Schneider. *Replication management using the state-machine approach.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993.

[45] J. Staten, S. Yates, G. F., W. Saleh, and A. Dines. Is cloud computing ready for the enterprise? Technical report, Forrester Research, Inc, 2008.

[46] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[47] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

[48] VMware Inc. VMware Server. http://www.vmware.com/, 2010.

[49] T. von Eicken. Amazon usage estimates, Oct 2009.

[50] Yahoo! Inc. Yahoo! Mail. http://mail.yahoo.com, 2010.

[51] D. Yang. *Java Persistence with JPA.* Outskirts Press, 2010.