André Frederico Guilhoto Monteiro

Gestão e engenharia de CAP na Nuvem Híbrida

HPC Management and Engineering in the Hybrid Cloud

André Frederico Guilhoto Monteiro

Gestão e engenharia de CAP na Nuvem Híbrida

HPC Management and Engineering in the Hybrid Cloud

Tese apresentada às Universidades do Minho, Aveiro e Porto para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Informática no âmbito do doutoramento conjunto MAP-i, realizada sob a orientação científica do Doutor Cláudio Jorge Vieira Teixeira, equiparado a Investigador Auxiliar, e do Doutor Joaquim Manuel Henriques de Sousa Pinto, Professor Auxiliar ambos do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Ao meu pai, o verdadeiro “engenheiro” que me inspirou na procura das coisas inovadoras, à minha mãe pela seu acompanhamento e exigência na educação e à mulher da minha vida pelo encorajamento, trabalho suplementar e paciência extra.

o júri / the jury

presidente / president

Prof. Doutor Domingos Moreira Cardoso
Professor Catedrático da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Fernando Manuel Augusto Silva
Professor Catedrático da Faculdade de Ciências da Universidade do Porto

Prof. Doutor Alfredo Moreira Caseiro Rocha
Professor Associado com Agregação da Universidade de Aveiro

Prof. Doutor Ignacio Blanquer
Professor Associado da Universidade Politécnica de Valência

Prof. Doutor José Miguel Oliveira Monteiro Sales Dias
Professor Associado Convidado do Instituto Universitário de Lisboa

Prof. Doutor Filipe João Boavida Mendonça Machado Araújo
Professor Auxiliar da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Prof. Doutor Nuno Manuel Ribeiro Preguiça
Professor Auxiliar da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Prof. Doutor Cláudio Jorge Vieira Teixeira
Equiparado a Investigador Auxiliar da Universidade de Aveiro

agradecimentos

Apenas umas linhas de texto não fazem jus ao sentimento de gratidão que tenho para com as pessoas que me ajudaram, nem conseguem albergar todas as que de alguma forma me conseguiram incentivar, agradeço a todas e em especial a algumas.

Ao Professor Joaquim Sousa Pinto, começando pelo convite para o Doutoramento, baseado na fé que teve em mim e nas minhas capacidades. Agradeço a confiança que sempre mostrou em mim e no meu trabalho e pela calma que sempre me transmitiu. Um exemplo, um incentivo persistente e a capacidade de ler nas entrelinhas e colocar os pontos nos "i"s onde faltavam, os pequenos detalhes que fazem um trabalho distinto.

Ao Professor Cláudio Teixeira, em primeiro lugar pelo desassossego de me tirar da minha zona de conforto e me lançar em novas tecnologias. O facto é que a investigação e a descoberta neste campo me fizeram abraçar com um gosto inimaginável à *Cloud*. Depois pela persistência e acompanhamento, a distância dos locais de trabalho foram-se encurtando com uma reunião, um correio eletrónico ou um telefonema. As correções e os comentários incisivos e frontais, que me obrigaram a moldar e aperfeiçoar o trabalho e a escrita. Desde Lodo tive a companhia deste Patrão, mais não podia desejar.

Agradeço à Universidade de Aveiro, e especialmente ao IEETA e ao DETI, mas também aos STIC a oportunidade e os recursos disponibilizados para a minha formação, humanos e materiais. Aos Professores cujas disciplinas pude dar aulas, pela oportunidade e aprendizagem, foi uma experiência fantástica.

Aos meus amigos, pela presença na minha vida e da minha família, que possamos continuar unidos e lembrarmo-nos sempre uns dos outros. Em especial ao meu irmão, que de longe se mantém bem perto.

À minha mãe, pelo apoio e incentivo ao longo destes anos mas sobretudo durante todo o percurso escolar, exigindo sempre o melhor de mim.

Ao meu pai, que mesmo não estando cá me proporcionou o melhor de uma vida e ao mesmo tempo me despertou o interesse pela tecnologia e engenhocas.

Ao Gustavo e ao Gaspar, que ainda não sabendo são uma fonte de motivação e alegria no final do dia-a-dia!

À Lena, mulher da minha vida e mãe dos meus filhos, não podia ter melhor amiga nem companheira! Obrigado pela infinita paciência e apoio intangível!

palavras-chave

Computação em Nuvem, Computação de Alta Performance, Nuvem híbrida, migração, CAP na Nuvem

resumo

O desenvolvimento e maturação da Computação em Nuvem abriu a janela de oportunidade para o surgimento de novas aplicações na Nuvem. A Computação de Alta Performance, uma classe dedicada à resolução de problemas complexos, surge como um novo consumidor no Mercado ao aproveitar as vantagens inerentes à Nuvem e deixando o dispendioso centro de computação tradicional e o difícil desenvolvimento em grelha.

Situando-se num avançado estado de maturação, a Nuvem de hoje deixou para trás muitas das suas limitações, tornando-se cada vez mais eficiente e disseminada. Melhoramentos de performance, baixa de preços devido à massificação e serviços personalizados a pedido despoletaram uma atenção inusitada de outros mercados.

A CAP, independentemente de ser uma área extremamente bem estabelecida, tradicionalmente tem uma fronteira estreita em relação à sua implementação. É executada em centros de computação dedicados ou computação em grelha de larga escala. O maior problema com o tipo de instalação habitual é o custo inicial e o não aproveitamento dos recursos a tempo inteiro, fator que nem todos os laboratórios de investigação conseguem suportar.

O objetivo principal deste trabalho foi investigar novas soluções técnicas para permitir o lançamento de aplicações CAP na Nuvem, com particular ênfase nos recursos privados existentes, a parte peculiar e final da cadeia onde se pode reduzir custos. O trabalho inclui várias experiências e análises para identificar obstáculos e limitações tecnológicas. A viabilidade e praticabilidade do objetivo foi testada com inovação em modelos, arquitetura e migração de várias aplicações.

A aplicação final integra uma agregação de recursos de Nuvens, públicas e privadas, assim como escalonamento, lançamento e gestão de aplicações CAP. É usada uma estratégia de perfil de utilizador baseada em autenticação federada, assim como procedimentos transparentes para a utilização diária com um equilibrado custo e performance.

keywords

Cloud Computing, High-Performance Computing, hybrid Cloud, migration, HPC in the Cloud

abstract

The evolution and maturation of Cloud Computing created an opportunity for the emergence of new Cloud applications. High-performance Computing, a complex problem solving class, arises as a new business consumer by taking advantage of the Cloud premises and leaving the expensive datacenter management and difficult grid development.

Standing on an advanced maturing phase, today's Cloud discarded many of its drawbacks, becoming more and more efficient and widespread. Performance enhancements, prices drops due to massification and customizable services on demand triggered an emphasized attention from other markets.

HPC, regardless of being a very well established field, traditionally has a narrow frontier concerning its deployment and runs on dedicated datacenters or large grid computing. The problem with common placement is mainly the initial cost and the inability to fully use resources which not all research labs can afford.

The main objective of this work was to investigate new technical solutions to allow the deployment of HPC applications on the Cloud, with particular emphasis on the private on-premise resources – the lower end of the chain which reduces costs. The work includes many experiments and analysis to identify obstacles and technology limitations. The feasibility of the objective was tested with new modeling, architecture and several applications migration.

The final application integrates a simplified incorporation of both public and private Cloud resources, as well as HPC applications scheduling, deployment and management. It uses a well-defined user role strategy, based on federated authentication and a seamless procedure to daily usage with balanced low cost and performance.

Table of Contents

List of Acronyms

API	*Application Programming Interface*
AWS	*Amazon Web Services*
CC	*Cloud Computing*
CDMI	*Cloud Data Management Interface*
CIMI	*Cloud Infrastructure Management Interface*
CMP	*Cloud Management Platforms*
CPU	*Central Processing Unit*
DB	*Database*
DHCP	*Dynamic Host Configuration Protocol*
DIMM	*Dual In-line Memory Module*
ECC	*Error-Correcting Code memory*
FB-DIMM	*Fully Buffered DIMM*
FCCN	*Foundation for National Scientific Computing*
GAE	*Google App Engine*
GASPI	*Global Address Space Programming Interface*
GFS	*Global Forecast System*
GPGPU	*General-purpose Computing on Graphics Processing Units*
HDFS	*Hadoop Distributed File System*
HPC	*High-Performance Computing*
HTC	*High-Throughput Computing*
IaaS	*Infrastructure-as-a-Service*
I/O	*Input/Output*
IEETA	*Institute of Electronics and Telematics Engineering of Aveiro*
IT	*Information Technology*
ITU	*International Telecommunication Union*
LXC	*Linux Containers*
MPI	*Message Passing Interface*
MS	*Microsoft*
NFS	*Network File System*
NIST	*National Institute of Standards and Technology*
NREN	*Europe's National Research and Education Networks*
OCCI	*Open Cloud Computing Interface*
OpenMP	*Open Multiprocessing*
OpenSHMEM	*Open Symmetric Hierarchical Memory*
OS	*Operationg System*
OVF	*Open Virtualization Format*
PaaS	*Platform-as-a-Service*
PC	*Personal Computer*
PGAS	*Partitioned Global Address Space*
QoS	*Quality of Service*
RAM	*Random Access Memory*
REST	*Representational State Transfer*
RHEV	*Red Hat Enterprise Virtualization*
RTC	*Real-time charging*
SaaS	*Software-as-a-Service*
SAML	*Security Assertion Markup Language*
SSD	*Solid State Drive*
SOA	*Service-Oriented Architecture*
SOCCA	*Service-Oriented Cloud Computing Architecture*
SSO	*Single Sign-On*
TOSCA	*Topology and Orchestration Specification for Cloud Applications*
UA	*University of Aveiro*
UDDI	*Universal Description Discovery and Integration*
VM	*Virtual Machine*
VSMP	*Versatile Symmetric Multiprocessing*
XML	*Extended Markup Language*
WPS	*WRF Preprocessing System*
WRF	*Weather Research and Forecasting Model*

List of Figures

List of Tables

Chapter 1

1 Introduction

1.1 Context

The computer industry is constantly evolving and with it all related background and habits. One of the deepest changes taking place is the migration from in-house computing services to the Cloud. In view of the fact that the Cloud reduces the difficulties in deployment of services and applications, for instance, Clouds are considered as a new generation of Grid computing [1]. Since its revival in the start of the XXI century, the Cloud concept has matured and evolved, but only on late 2008 had it exponentially grown and by 2010 it was the number one buzzword in IT industry. Figure 1.1 shows the evolution over time of news headlines for the search terms “Cloud computing” and “Grid computing” on Google Trends [2], which had an exponential growth from 2008 to 2012.

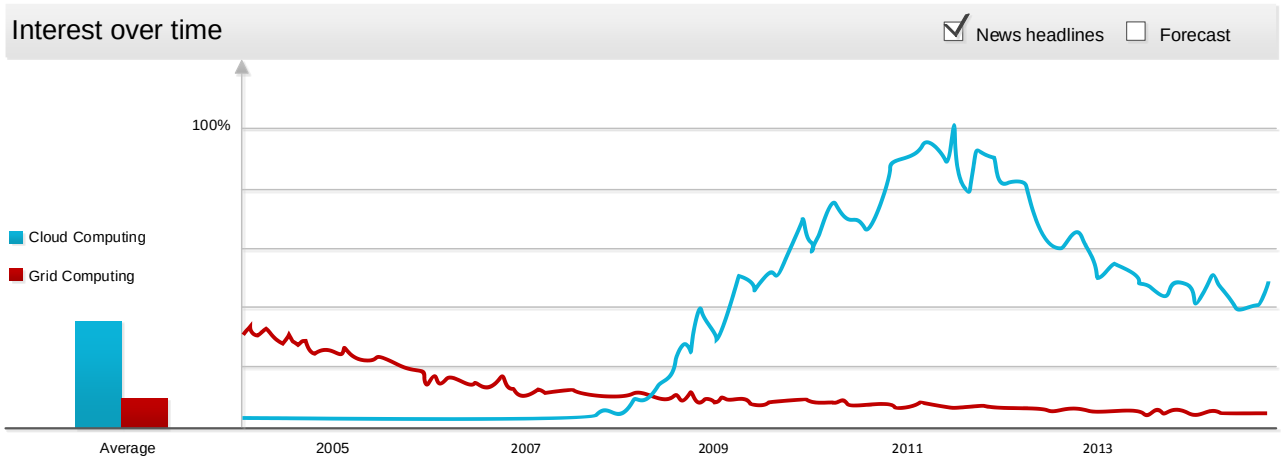

Figure 1.1 - Interest over time on Cloud and Grid computing topics

The concept of Cloud Computing started back in the 1960's, when John McCarthy publicly suggested that computer time-sharing technology might lead to a future in which computing power and even specific applications could be sold through the utility business model (like water or electricity). The idea of an "intergalactic computer network" [3] was introduced in the fall of the same decade by J.C.R. Licklider, responsible for enabling the development of ARPANET (Advanced Research Projects Agency Network). Virtualization, a key concept that comes along with Cloud Computing, was proposed by Christopher Strachey as “time-sharing” and implemented on Atlas and IBM M44/44X machines [4] in the same period. Figure 1.2 depicts an overview of the Cloud Computing timeline.

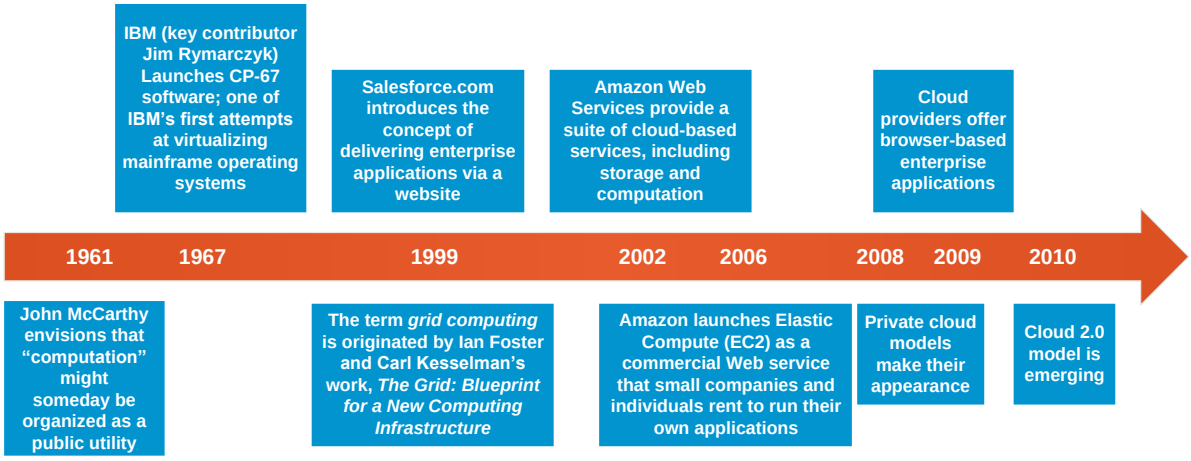

Figure 1.2 - Cloud Computing timeline [5]

Cloud Computing can be defined as the collection of scalable, virtualized resources, which is capable of hosting applications and providing required services to the users. In Cloud Computing, software, hardware and network play the main role. The collective efforts of these entities make Cloud Computing possible. We can visualize the Cloud as a cluster of connected computers which are based upon distributed systems that provide services in real time. There are several reasons why companies are moving IT services to the Cloud. Cost is clearly the first; the second is functionality. In most cases, Cloud providers can offer a more cost-effective and robust service than in-house solutions. These datacenters have high costs and needs like maintenance, networking, floor space, cooling, power and disaster recovery strategies.

Grid Computing and Cloud Computing are often mistaken. Grid computing is by definition a linking of several computers in parallel to run specific applications or solve certain problems. On the other hand, Cloud Computing represents the aggregation of multiple resources, as CPU, RAM and storage, which delivers a service to the user. A grid is a system that has the ability to manage and organize resources and services [1], using protocols and interfaces to distribute tasks. Grid Computing usually can divide parts of a single large image, dataset or process to several thousand computers, remotely distributed. Then each computer runs a daemon to process its part and returns the result to the main node, which assembles the resulting image. The fact is that they are somewhat alike and that is why this perception exists. Both Grid and Cloud Computing are scalable, allocated/de-allocated on demand and multitenant, clients can perform different tasks, accessing a single or multiple instances. However, Cloud Computing provides services at several abstraction levels and encloses a whole set of different providers, operating systems, applications and users. We can even see Cloud Computing as an evolution of Grid, with Cloud being service-oriented and Grid Computing application-oriented.

High Performance Computing (HPC) [5] stands for intensive computing in high throughput hardware systems. It is a multidisciplinary area that blends programming techniques, as parallel programming, with powerful hardware systems to accomplish tasks that would be unfeasible with regular systems. HPC combines the power high-performance computing infrastructure with highly efficient programming solutions to solve scientific and data analysis problems. Traditional HPC is performed on large dedicated datacenters, owned by prevailing enterprises, governments or universities, due to its heavy cost. It can be carried out by infrastructure owners or lent to outside clients in a business model perspective.

Nowadays, HPC in the Cloud has become one of the most resource-effective ways to compute data. Without spending all the funding in large dedicated datacenters, which require a big initial investment and expensive maintenance, HPC tasks can be processed and delivered in a very efficient and flexible way. These capabilities enable innovation by minimizing the need for innovators to find resources to develop, test, and make their innovations available [1]. Distributed computing

processing as a service has become a cutting edge technology: data can be processed and returned seamlessly within similar requirements, as time and money.

The main problem is that HPC cannot simply be moved out from traditional datacenters to Cloud Computing. There are several constraints, mainly technological, that make this porting a reasonably hard process, as the lack of standards. While searching the ideal solution for the Cloud portability, others leverage solutions, as wrappers, Application Programming Interface (API) or middleware, for the increasing HPC requests and needs.

While standards and process generalization take time, this has been the easiest way of quickly taking advantage of HPC on any Cloud. Creating a wrapper, for instance, establishes a new layer to the user, seamlessly linking functionality to the lower layers. APIs are also very common, Cloud providers grant an alternative way of manage resources implemented, for example on Representational State Transfer (REST).

Middleware software acts as an interface between the client and the Cloud servers, and provides a service in a transparent way. Building a platform independent Cloud application that is efficient, transparent and secure to launch and access applications across different operating system environments and over a network of Cloud servers is not an easy task. There are many levels to evolve in an always changing heterogeneous environment as the Internet. The singular Internet environment induces to unpredictable product complexity on software development and constant requirement changes.

Such products require agile software development approaches that balance flexibility and disciplined methodology to achieve success.

1.2 Motivation

HPC and Cloud Computing have exponentially evolved in the last few years. To compete in the digital economy, companies must be able to develop high quality software systems at “Internet speed” — that is, deliver new systems to customers with more value and at a faster pace than ever before. The Internet environment intensifies software development problems by emphasizing shorter cycle times [6]. Pressured by time and with an ever increasing focus on managing expenditure, today’s clients analyze online providers to offer maximum choice, at the most competitive price [7]. Within this context there is the need to offer a solution to the market demand, especially taking advantage of the Cloud vicissitudes. How do HPC applications fit on the Cloud framework? How can they be deployed? From the customer’s point of view, prepaid services may seem to be resource wasteful for the client when all contracted usage is not spent. Likewise, postpaid services may also seem to lack flexibility to change in the middle of the billing period, and limit applications. When it comes to the Cloud, several key features pop up, as elasticity. With the pay-per-use model, one can maintain its servers to a minimal configuration, at a very low price, and scale up easily in minutes when demand calls.

At this point new business models are set to emerge. There are several application areas ready to widen their scope of appliance, and due to technology evolution it is set to be the future. To make these business models possible, Cloud Service Providers should enforce more effective handling of settlement between multiple parties in the service delivery chain. Real-time charging (RTC) is being pointed out as a potential solution to encourage greater personalization and niche service delivery to targeted users [7].

On the other hand, almost completely aside from the service and technology development process, end-users spend more time online every day. The normal usage of the Internet, such as social networks, blogs and news reading, use low-resource consuming applications, wasting the PCs capabilities, everyday more powerful. Why not share users PCs’ resources to provide services to others and eventually have some kind of revenue [8]? This approach opens the possibility of the

exploitation of the joint end-user resources, both home and enterprise. Aggregated, they combine very powerful computing resources.

All these new technologies present a unique opportunity for development departments to leverage IT innovations. These innovations open up new opportunities for investigation and process improvement. For instance, in a near future, researchers can manage powerful research facilities on web-based forms, instantly aggregating results, rankings and comments into a shared, securely viewable data. This solution not only minimizes costs but also shortens the time from concept to launch and significantly streamlines internal collaboration and result sharing.

In order to benefit from these new models, either for providers or end-users, it is necessary to study market needs and deliver a software implementation that fulfills all requirements and makes use of the existing and underused infrastructure and platforms. How to merge these technologies to make the best out of resources?

In the years to come, HPC within the Cloud will take a substantial part of the IT landscape, replacing traditional deployment models as generic clusters, able to support a wider range of applicability. This implies a growing demand for a Cloud environment that can operate parallel applications, thus saving resources and money. Yet, the application of HPC in the Cloud has just started and will take a couple more years to mature [6]. Main problems include performance issues (HPC wise) and data reachability. An additional issue is the cost of utilizing the Cloud resources themselves. One way of overcoming the potential cost barrier is to take the maximum advantage of existing on-premise computing resources and, when needed, reach out for external resources. There are also other ways to balance costs and take advantage of the HPC in the Cloud [9][10], but there is no easy comparison as the Cloud poses several intrinsic characteristics.

1.3 Objectives

This thesis aims to prove the feasibility of HPC over a hybrid Cloud, applying a coordination theory perspective to a solution deployment. It is also intended to present a conceptual framework and methodology for analyzing coordination and cooperation requirements in HPC project tasks over this Cloud infrastructure.

In this approach, the combined usage of private infrastructures and third party infrastructures provides a cost-wise balanced solution. On one hand, local processing resources are better used; on the other hand, a public Cloud is always updated with the state-of-the art hardware, available 24/7 with a couple of minutes' deployment and just pay as you go. The main challenge is to make HPC feasible in this hybrid scenario, using a Cloud of Clouds, both private and public, to accomplish a balanced and useful application for the research community.

More specifically, this thesis aims at:

- Bringing innovative technical solutions on the Cloud for research applications;
- Leveraging workload mobility within the hybrid Cloud;
- Developing a small budget solution for intensive computing;
- Providing an agile way to manage all the gathered resources.

1.4 Major contributions

This document includes proposals to achieve a better alignment between all involved technologies and their implementations, for a wide field of practices. Simultaneously, it shows how to move towards a business coordination support and inter-organizational collaboration, with a high level of abstraction, using the existing APIs and open-source software to build applications on the top layer, modular and open to extensions for other important research work. This is an important issue due to peculiar settings in each research area that must be compatible to fit the Cloud.

These are the main contributions to this scientific area:

- A highly configurable architecture model which can sustain a wide range of future applications;
- Extension of HPC projects into the backbone of some of the largest data computing infrastructures;
- Dynamic and self-provisioning of systems tied directly to workload requirements;
- Common management of on-premise business, compute and Cloud systems;
- Development of a highly elastic, powerful yet small budget solution for HPC problems;
- Building of a hybrid Cloud Computing network;
- HPC experience with several projects, application reengineering and result analysis.

1.5 Thesis organization

This document is organized in 7 chapters, starting with a contextualizing summary and motivation on chapter 1, followed by 3 chapters with a comprehensive state-of-the art of the concepts and technology used – from High-Performance Computing, Virtualization, throughout Cloud Computing and ending in HPC in the Cloud – traditional, Cloud and hybrid. Starting from chapter 5 and ending in chapter 6, the performed work is fully detailed, as well as the considered options and walkthroughs. The thesis ends with chapter 7, where the research work is evaluated and analyzed, including future considerations and related contributions.

Chapter 2

2 Virtualization

Virtualization has been present for a long time, although it may seem to be a new and revolutionary technology. As shown back in Figure 1.2, it appeared in the 60s decade as a way to improve mainframe hardware utilization, and has evolved alongside the recent Cloud exponential development. After decades being disregarded, the resurgence of interest in virtualization was led by the capacity of current PCs to present an illusion of smaller Virtual Machine (VM), each running a separate OS [11].

Although it is a very broad concept, virtualization can be defined as an abstracted, virtual view of the hardware to guest operating system instances, which run on a single physical machine [12]. This is accomplished by the hypervisor, a central item that runs the virtualization layer directly over the system, rather than accessing resources through an OS. The hypervisor implements the virtual machine hardware abstraction and is responsible for running the guest OS.
Virtualization also has drawbacks, as it would be expected. Some pros and cons of virtualization can be overseen in Table 2.1.

Table 2.1 - Virtualization pros and cons

Advantages	It can make use of virtual machine templates, with fast deployment and configuration
	Use more of the computer's processing and memory resources than servers running a single OS instance and a single set of services
	Supports better control of OSs to ensure that they meet the organization's security requirements
	Peak workload can go beyond what standard cluster provides, with load-aware scheduler
Disavantadges	Virtualization adds layers of technology/logic, which increases performance impact
	Conflicting schedule policies among different groups
	Combining many systems onto a single physical computer can cause a larger impact if a security compromise or a hardware failure occurs
	Virtualized environments are quite dynamic, which makes creating and maintaining the necessary security boundaries more complex

The main achievement is fast deployment and increased capacity by shared resources; the major drawbacks are performance and security management.

Although there are several technical differences between processor manufacturers, Intel and AMD have hidden the implementation details and provided seamless access to hypervisor support. The hypervisor support can be basic or have advanced features, depending on the processor model.

Virtualization can be used to accomplish many objectives like increasing efficiency. By virtualizing their workloads, companies improve scalability and flexibility while reducing costs. Nowadays, almost every medium-budget CPU has hypervisor support directly embedded in the hardware, capable of managing and sharing physical resources. As not all computers are high resources consumers, each computer is able to share the unused resources.

The increasing development of virtualization has brought distributed systems several benefits [13] such as:

a) server consolidation, allowing workloads of several under-utilized servers to be placed in fewer machines;
b) the ability to create VMs to run legacy code without interfering in other applications' APIs;
c) improved security through the creation of sandboxes for running applications with questionable reliability;
d) dynamic provision of VMs to services, allowing resources to be allocated to applications on the fly;
e) performance isolation, thus allowing a provider to offer some levels of guarantee and better quality of service to customers' applications.

2.1 Types of Virtualization

The increasing search for the use of the virtualization model has been motivated by its many benefits, despite its abstraction costs. However, virtualization is more complex and is divided in functional groups, with different views and applications. Some of the most used types, full virtualization and paravirtualization, are described in the next paragraphs. There are also other types as partial, emulated, hardware assisted, OS level, application and cross-platform virtualization, among other subtypes.

Full Virtualization

Full virtualization presents a faithful emulation of hardware interfaces to guest OS which do not realize themselves as living in virtual machines and requires no change to the guest [14]. OS calls are caught using binary translation, guest OS do not have direct access to physical hardware. Systems can have specific software and configurations which could be replicated to a single image disk file and easily cloned. Figure 2.1 shows a full virtualization model overview:

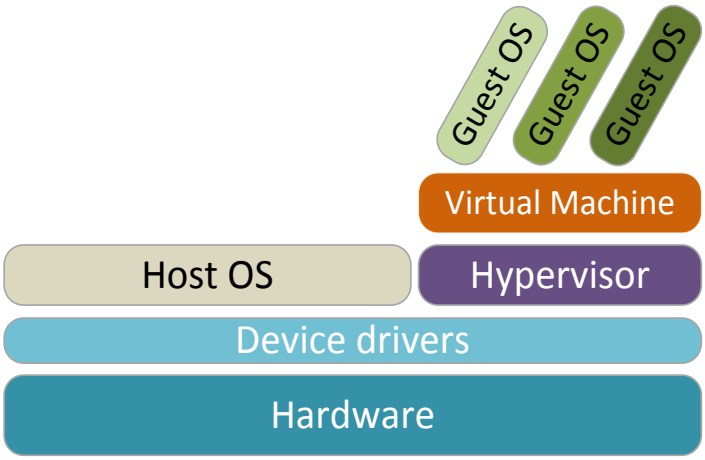

Figure 2.1 - Full virtualization model

One of the most common reasons for adopting full virtualization is operational efficiency: organizations can use their existing hardware more efficiently by putting more load on each computer [15], usually not fully used or exploited.

Paravirtualization

Paravirtualization delivers a VM abstraction similar, but not equal, to physical hardware. It has a specific method, a hypercall interface which allows guest OS to perform privileged operations through the hypervisor. For this reason, guest OS need some modifications while migrating into VM. The added value of paravirtualization is in lower overheads and consequently more speed [11]. However the performance advantage of paravirtualization over full virtualization depends on the workload and is not guaranteed on all tasks. Compatibility and portability is low, for instance standard OS are not supported. Paravirtualization can be illustrated as in Figure 2.2.

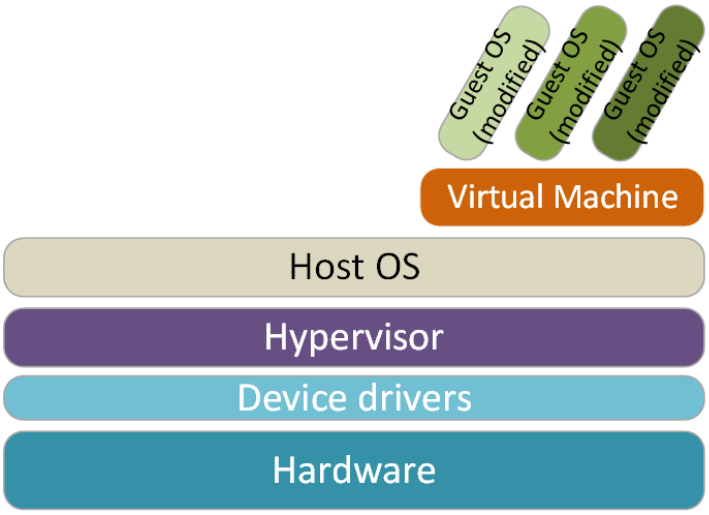

Figure 2.2 - Paravirtualization model

Sitting right on top of hardware, the type 1 hypervisor has inner contact with hardware, and thus has more privileges to execute operations. The problem, as referred before, it that every guest OS needs to be adapted to the hypervisor/hardware combination.

Container-based virtualization

A completely different approach is a container-based virtualization, which does not use a kernel at all. Linux Containers (LXC) is a userspace interface for the Linux kernel containment features, allowing to create an environment as close as possible to a standard Linux installation but without the need for a separate kernel [16]. Figure 2.3 shows the LCX model running guest OS under individual containers beneath the host OS.

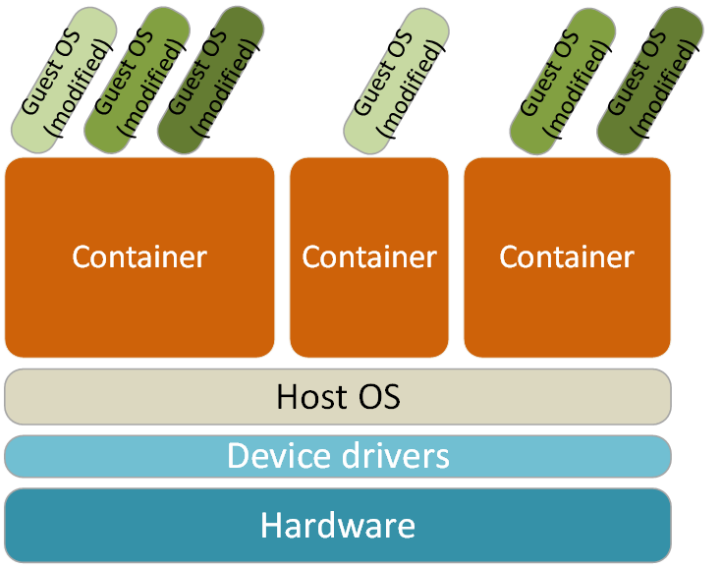

Figure 2.3 - Container-based virtualization

On this operating system–level virtualization, users can to create an environment similar to a normal Linux system without the need for a separate kernel, called containers. Rather than using a fraction of the machine's physical resources as hypervisors do, LXC is able to isolate processes and allocate resources without the use of hardware emulation. This can increase application portability as well as physical resource usage. On some comparisons [17] LXC has shown very good performance for several applications, mainly reducing deployment time and showing a good flexibility.

2.2 Hypervisors

The hypervisor is the central piece of virtualization, and is the major responsible for almost all operations and management. When the system boots, the hypervisor runs, reads the configuration files and takes control of devices and disks. Once the process is up and running, all further operations can start. Despite its low success in the 60's mainframes, in 1999 VMWare appeared with a hypervisor for x86 systems, which could virtualize hardware resources for a wider market. This innovation was a trigger for virtualization development, both on hardware and software. Then came the Cloud as we know it.

Hypervisors divide in two basic types: type 1 and type 2. Type 1 sits directly on the host hardware while type 2 runs over a host OS, as illustrated in Figure 2.4.

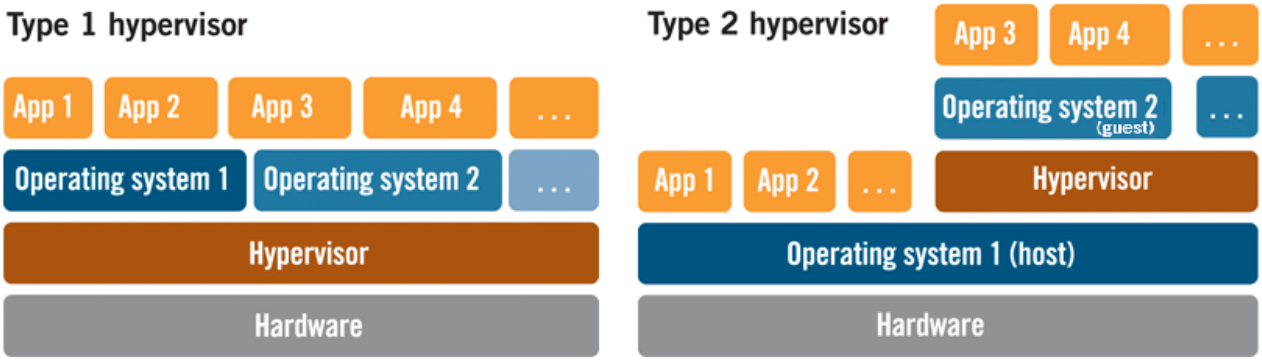

Figure 2.4 - Hypervisor types

The key dissimilarity is the existance of a host on type 2 and modified guest OS on type1. There are several hypervisors in the market, some of them more used than others. There are licensed hypervisors and open-source hypervisors, much common these days and with increasing use. Figure 2.5 shows a recent (2013) market survey [18] indicating the current market share for OpenStack users.

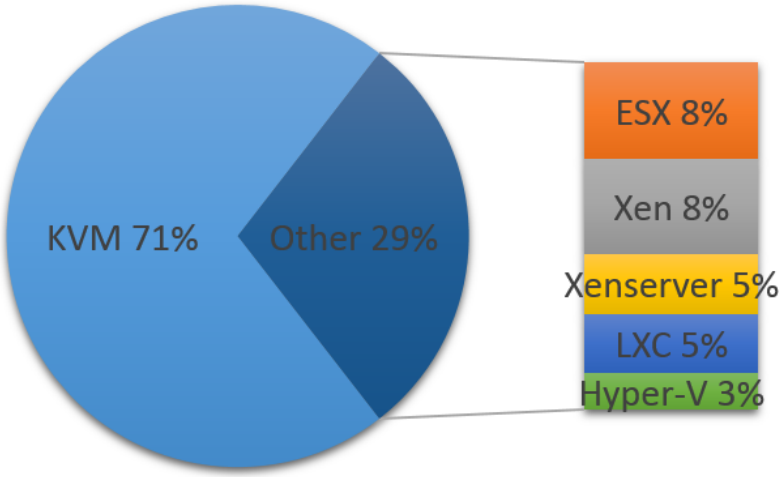

Figure 2.5 - Hypervisor market share [18]

The open-source hypervisor Kernel-based Virtual Machine (KVM) leads the user preferences, followed by VMWare ESX and Xen. Some of the most important hypervisors are presented on the following sections, followed by a comparison table to illustrate the main differences.

KVM

KVM is a full virtualization solution for Linux on x86 architecture hardware containing virtualization extensions (Intel VT or AMD-V). It consists of a loadable kernel module that provides the core virtualization infrastructure and Intel/AMD processor specific module [19]. KVM is a type 2 hypervisor and because it was designed after the emerging of hardware assisted virtualization, it did not implement features that were provided by hardware. Thus, the KVM hypervisor requires Intel VT-X or AMD-V enabled CPUs and leverages those features to virtualize the CPU.

One of the advantages of KVM is that by requiring hardware support rather than optimizing hardware, its solution does not require legacy hardware support and modifications to the guest operating system.

KVM is one of the mostly used hypervisors, is free to use and, thanks to its incorporation in Linux kernel 2.6.20, it was largely distributed from 2007. Figure 2.6 depics KVM architecture, which presents a virtualization module inside the OS kernel.

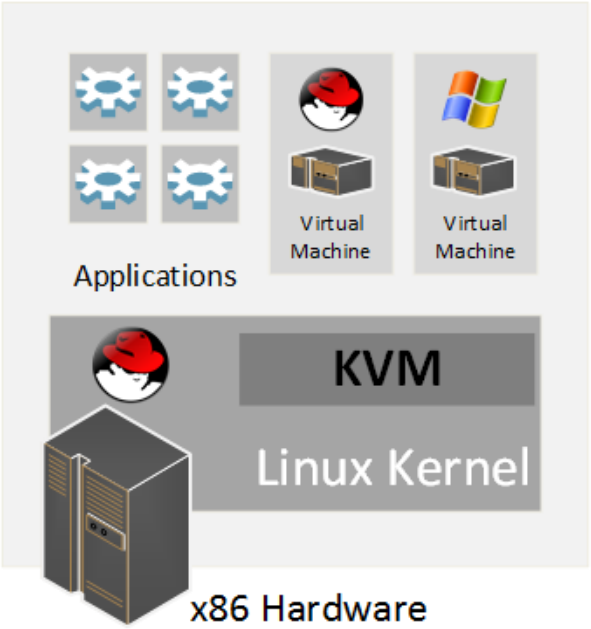

Figure 2.6 - KVM architecture [19]

KVM, embedded on a Linux OS (still no Windows version is available), has a typical type 2 hypervisor architecture, lying on x86 hardware and allowing regular applications to run with VM side by side.

Xen

Xen is a native hypervisor running directly on the host's hardware, which can provide a paravirtualization or a full virtualization. It is also an x86 architecture virtualization solution founded over Intel VT/AMD-V virtualization extensions on a Unix host [20].

It started as an open-source project at the University of Cambridge, with the first release in 2003. It evolved as one of the leading hypervisors used and in 2007 it was bought by Citrix Systems, which used it to build several business applications around it (XenExpress, XenServer, XenEnterprise), though it still managed to stay open-source.

The Xen is a type 1 hypervisor which provides a platform for deploying a wide variety of network-centric services, such as local mirroring of dynamic web content, media stream transcoding and

distribution, multiplayer game and virtual reality servers, and smart proxies [11]. Xen architecture is illustrated on Figure 2.7.

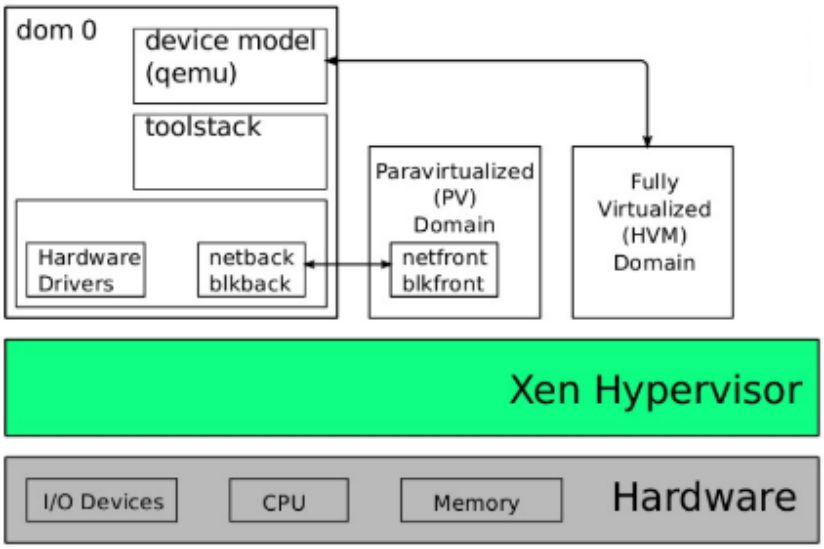

Figure 2.7 - Xen architecture [20]

Being a type 1 hypervisor, Xen is positioned right above the hardware, allowing services and VMs to run. A peculiarity is the deployment of dom0, a single privileged virtual machine that has direct access to hardware.

LXC Docker

Altough LXC is not a hypervisor, its virtualization capabilities and flexibility have drawn the attention of many vendors lately, including a fast-growing community. Within 2014, Docker [21] has emerged as a standard runtime, build system image and format for LXC. Docker is a toolset that makes it easy to package an application and all of its dependencies into a container. LXC provide lightweight guest OS virtualization, it runs under the OS and consumes only the required RAM and CPU, instead of pre-allocating it. Figure 2.8 depicts Docker architecture, running inside the host OS.

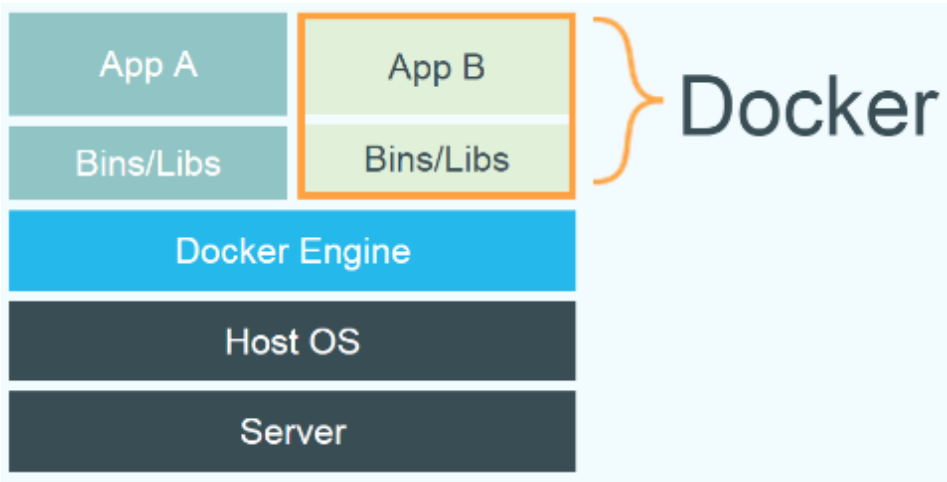

Figure 2.8 - Docker architecture

Unlike hypervisors, LXC does not require hardware architecture support. A container basically allows processes and their resources to be isolated without any hardware emulation or hardware requirements thus providing a platform where each container runs their own OS but share the host kernel. Each container can run its own Linux distribution, has its own filesystem (Advanced Multi-Layered Unification Filesystem - AUFS) and network. These abstractions make a container behave like a VM but with no hardware emulation at all.

ESXi

VMware ESXi is a next-generation hypervisor, providing a new foundation for virtual infrastructure. This innovative architecture operates independently from any general-purpose operating system, offering improved security, increased reliability, and simplified management. The compact architecture is designed for integration directly into virtualization-optimized server hardware, enabling rapid installation, configuration, and deployment [22].

ESXi is a type 1 hypervisor and has a free version, which has basic features along with a memory limitation (maximum 32GB) and includes a management tool and storage/memory optimization.

Figure 2.9 shows the implemented architecture. The hypervisor has a VMkernel that contains a range of subfunctional blocks to manage VM, services and client interfaces; the VMs are deployed above this layer.

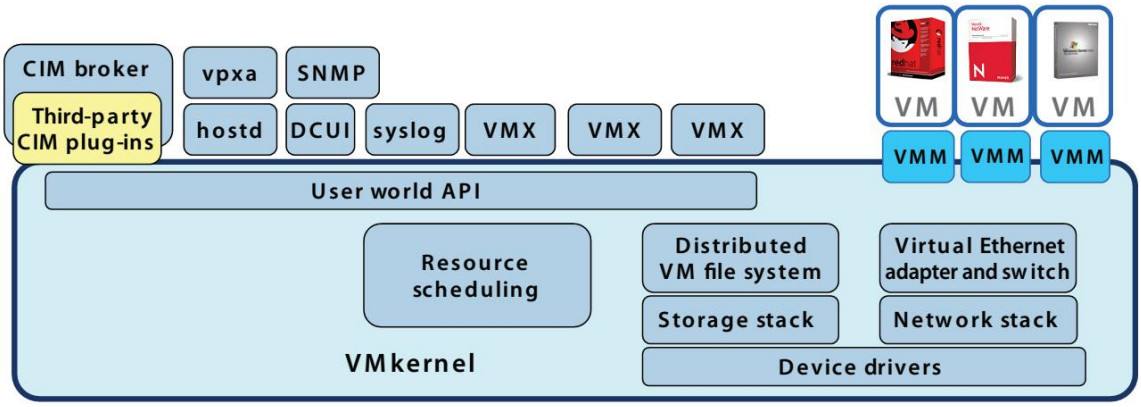

Figure 2.9 - ESXi architecture [22]

Hyper-V

In the field of hypervisors, Microsoft has also entered the market with Hyper-V in 2008. However and although it is a type 2 hypervisor, it is embedded to Windows Server Hyper-V, a deployment model somehow different than usual. Once Windows Server is installed, Hyper-V is installed as a role. When this finishes, Hyper-V is installed into the kernel, directly on top of the hardware, and the Windows instance becomes a parent partition. Figure 2.10 shows how the Microsoft model is designed.

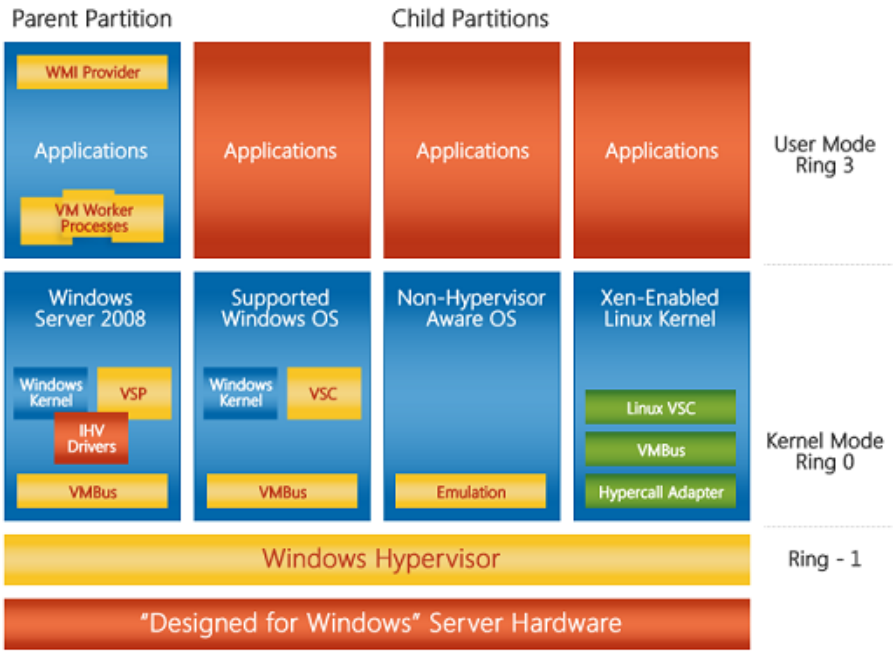

Figure 2.10 - Hyper-V architecture [4]

Microsoft has separated the model in two logical parts, so called partitions. The parent partition has the main OS, which controls and manages the available resources and requests directly over the hypervisor. The child partitions are all guest OS, with a note to other OS rather than Windows – they only run with emulation.

Comparing Hypervisors

These hypervisors are available on the market, each one with their own characteristics, which can be more suitable for some users than others. It keeps depending on its main use, target application, legacy system, VM desired platform, legal issues and of course budget, among others. Some of the most important features are detailed on Table 2.2.

Table 2.2 - Hypervisor comparison

Hyper visor	Type	Host OS	Max CPU	Max RAM	License	Supported guest OS	Supported images
KVM	2	Linux	160 cores	2TB	Open-source	Linux, Windows, BSD, MacOS, Haiku, Plan9, AROS, ReactOS	raw, qcow, vmdk, vdi, vpc
Xen	1	Linux / Windows	64 cores	1TB	Open-source	Linux, Windows, BSD, Solaris	raw, vhd, qcow, ovf
ESXi	1	Linux / Windows	160 cores	2TB	Free	Linux, Windows, MacOS X10	raw, vmdk, ovf
Hyper-V	2	Windows	320 cores	4TB	Commercial	Linux, Windows	vhd, ovf

Besides being a free and open-source hypervisor, KVM also benefits from being a type 2 hypervisor, meaning that no extra work has to be done to guest OS to run over it. This is one of the keys of having established itself as a market leader on non-paid solutions.

Chapter 3

3 Cloud Computing

The term Cloud Computing is very abstract and can compromise numerous components as services or resources. Although authors vary on definitions, one of the most widely used is the one created by NIST (National Institute of Standards and Technology). NIST has defined it in 2009 and updated its definition in 2011 to a model for enabling ubiquitous, 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 [7]. This definition also encompasses characteristics, service models and deployment models which are considered essential. These characteristics, on demand self-service, broad network access, rapid elasticity, resource pooling and measured service are explored on the following sections. Figure 3.1 shows the NIST model which comprehends deployment, service and intrinsic characteristics.

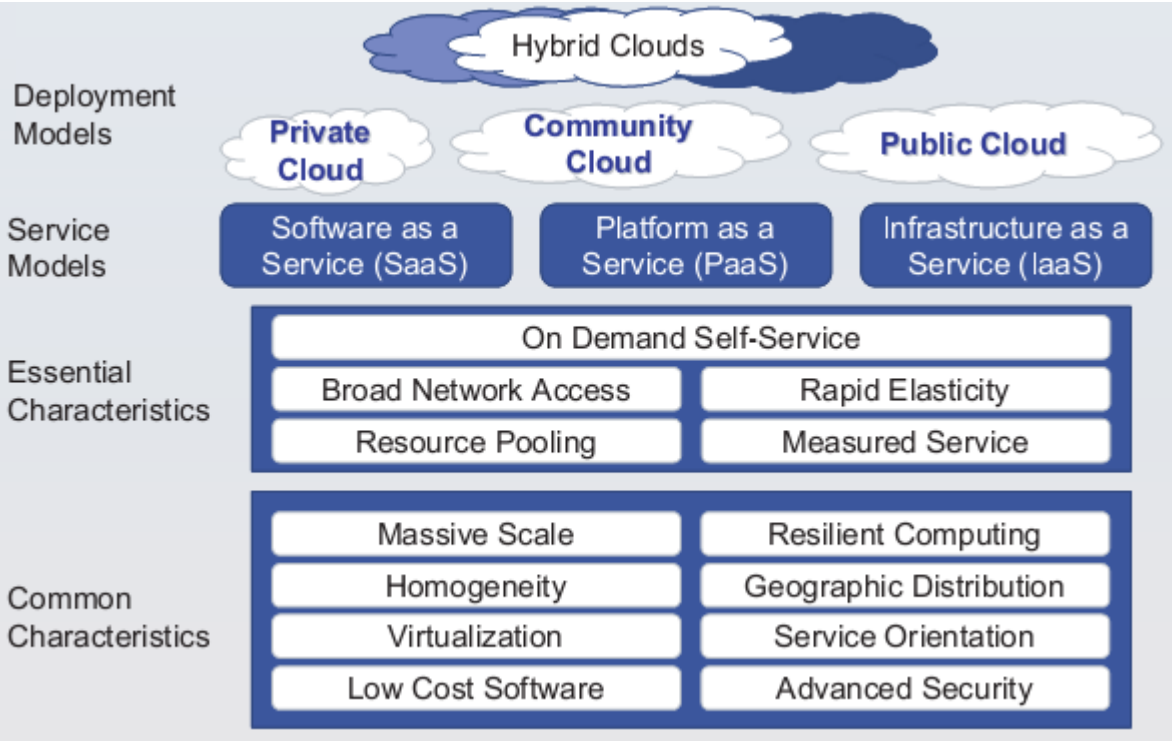

Figure 3.1 - NIST Cloud definition model [23]

Cloud Computing overlays some of the concepts of grid, distributed and utility computing, but fundamentally, Cloud Computing is a virtualization of resources (computing capacity, network and storage) with the ability to provide dynamic and scalable means to user's needs.

Traditional datacenters require a fair budget to keep applications and data equally running and secure. In a recent past, companies would only execute their applications on their own servers, hosted at these closed datacenters. Yet, the Cloud brought a different business perspective which complemented the existing distributed architectures implementations. Companies can now move much of their applications to Cloud services at a stunning speed and lower cost, and benefit from flexibility to scale when peak demands exist.

3.1 Types of Cloud

3.1.1 Public Cloud

The emergence of Cloud as we know it has begun when Amazon and others started to offer their computing resources publicly; this occurred in the middle of the dot com bubble burst, an internet boom development that ended in a stock market crash. In the case of Amazon, such offering was justified following a need of better usage of its owned servers' capacity, full on holiday seasons and nearly empty the rest of the year, Amazon launched the resource rental model, made it viable and commercially interesting. The public label means it is accessible by any person or company at any network connected location for a certain price.

3.1.2 Private Cloud

Private Clouds refer to Clouds with strict access, inside a single organization (company, lab and university, etc.), self-deployed and managed. They appeared later on stage with several small projects to fulfill some particular needs, like privacy, location and security, which users could not trust with public providers.

3.1.3 Community Cloud

A community Cloud comprehends characteristics of both private and public models. The infrastructure is used by a group of organizations that share common interests and manages it to serve their mutual purposes. This model can represent higher cost savings than the private Cloud while offering some of its security features and combined capacity.

3.1.4 Hybrid Cloud

The hybrid Cloud Computing model is a common model deployment within a large organization because it features aspects of all of the other models. An organization may use internal resources in a private Cloud maintaining full control over its sensitive data, but also process tasks internally with less cost. It can then use a public Cloud provider when scaling up for computing resources when demand overcomes internal capacity. Simultaneously, it can share resources with other organizations with similar interests. All this makes the hybrid model the most agile and flexible.

3.2 Cloud Computing Architecture

Cloud Computing layers are well defined and tend to separate roles clearly. Within each level, specific tasks are allocated and implemented. Figure 3.2 shows the Cloud layers common assignments and roles.

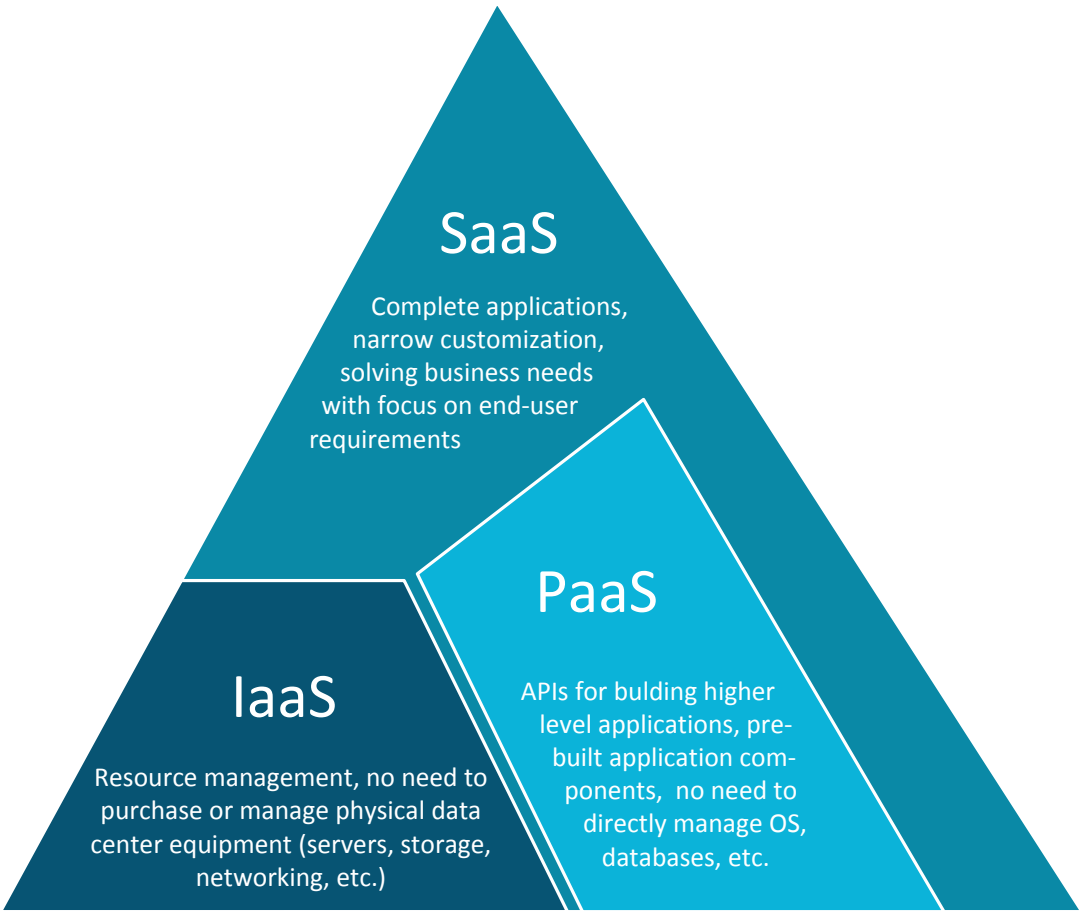

Figure 3.2 - Cloud Computing Layers

Each layer has a precise purpose: Infrastructure-as-a-Service (IaaS) manages and maintains physical resources, Platform-as-a-Service (PaaS) provides pre-built applications and Software-as-a-Service (SaaS) serves the end-user, delivering software and services.

3.2.1 IaaS

The infrastructure layer rests over the hardware and is capable of managing resources and a hypervisor which provides resource sharing and virtualization. This allows IaaS to offer users virtual machines on-demand as a service. Instead of acquiring physical servers, data storage systems, or networking resources, the client users just pay the used resources thanks to account and billing features of this layer.

On IaaS, users can provision and spawn an infrastructure or/and run certain software, with the particularity of its elasticity – it can grow or shrink according the workload variations. There are also three types of IaaS categories: public, private and hybrid. Public IaaS are business model services which publicly provide resources at their organization for a price. Private IaaS are software implementations that allow the users to use on-premises resources and manage their lifecycle. Hybrid IaaS combine the resource origin, they provide a mean to manage existing resources with some public provider, usually through APIs.

3.2.2 PaaS

Typically, the platform layer builds on the infrastructure layer's machine. At this level users can customize virtual machines it provides with any software package that end users may require, exposing only the APIs required to get the job done [24]. This is quite interesting for some users, because the user doesn't need to buy licenses or applications, they just need to use the platform and disconnect when finished. A key feature of PaaS is multitenant architecture, where several

unrelated applications run on the same hardware and software infrastructure. Unaware of deployment and other issues, developers can focus on the application itself.

Public PaaS were the first to appear on the market. As in the global Cloud concept, the public attribute refers to an accessible pay-per-user PaaS, offered by a public Cloud provider. The offering of this service aimed at developers as its main target, providing a set of application measuring, monitoring and deployment tools. Examples of public PaaS are Amazon Elastic Beanstalk [25], CloudBees[26], Google App Engine [27], IBM SmartCloud Application Services [28], Jelastic [29], Microsoft Azure [30], OutSystems Agile Platform [31], Red Hat OpenShift [32], Salesforce.com [33], Heroku[34], CloudFoundry [35] and WSO2 Stratos/StratosLive [36].

The first developments in the private Cloud started from the lower layer IaaS, as a bottom-up approach. As the first private Cloud implementations were steady and stable, developers turned their minds to private PaaS. One of the main objectives of many of these private PaaS projects was to achieve control over the bursting of virtual machines and their underlying virtual networks whose management was completely out of sight on commercial Clouds.

Currently, and while still maturing, the private PaaS are used to improve resource deployment, through more customized multitenancy and elasticity, improve the flexibility of the computing infrastructure, by abstracting the shared resources to higher levels of software stacks and, of course, cost reduction. Some of the most popular private PaaS are ActiveState Stackato [37], Cumulogic [38] and GigaSpaces Cloudify[39].

3.2.3 SaaS

The SaaS level consists in providing on-demand applications over the web, such as email, office and database applications, which do not require any development. Among SaaS characteristics are multitenancy, easy customization and better user configuration. SaaS completely abstracts the underlying mechanisms from the consumer; the system works as if a large computer were servicing all user requests. While SaaS is often offered directly to end-users, such as email, enterprise services can be charged periodically or by usage.

3.3 Infrastructure-as-a-Service layer

IaaS is the lower level layer of Cloud Computing, thus providing the most wide-ranging control over it. Here the user is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software [23]. The underlying Cloud infrastructure is not entirely managed or controlled by the user but the user can exert control over operating systems, storage, deployed applications, and possibly limited control of selected networking components as firewalls.

This is a very important layer, because it widens the window of customization and is the foundation for other layers' services. At this point, users can manage hosts, VMs, images, templates, virtual networks and user/groups. Entire workloads of VMs, each on with their own OS, can be deployed in a convenient way to fulfill their objectives.

For achieving the objetives of this work, it was imperative to work at the IaaS level. As early studies concluded that Clouds were not originaly designed for running tightly-coupled HPC applications [40], only a low-level development approach (redefining architecture, upgrading network performance and enhancing hypervisor, etc.) could create a feasible solution.

A set of the most popular IaaS providers, with free, commercial and open-source versions, is presented on the next sub-sections.

3.3.1 Amazon Elastic Compute Cloud EC2

Although not a free or open-source IaaS, Amazon EC2 presence is totally justified as it is one of the original creators of the Cloud itself. As already stated on 3.1.1 the EC2 platform was created to fulfill Amazon's capacity, originally bought to sustain one of the world's largest retail store, and with a lack of use out of special seasons. By developing the Public Cloud concept and without any standard constraints, Amazon established itself as a *de facto* standard, either on concept, modeling and resource managing. While maturing, both Private and Public Cloud clients would also benefit compatibility with EC2, either on image format or API management, leading Amazon to a solid rock position.

Amazon EC2 has evolved wisely and consistently, and offers plenty of service at the IaaS level. Besides the traditional VM lifecycle management through a web portal, EC2 also offers ready templates, storage management, privacy settings, custom security and spot-prices, which work like an auction. EC2 implementation is not available for public disclosure, but can be regarded [41] like Figure 3.3.

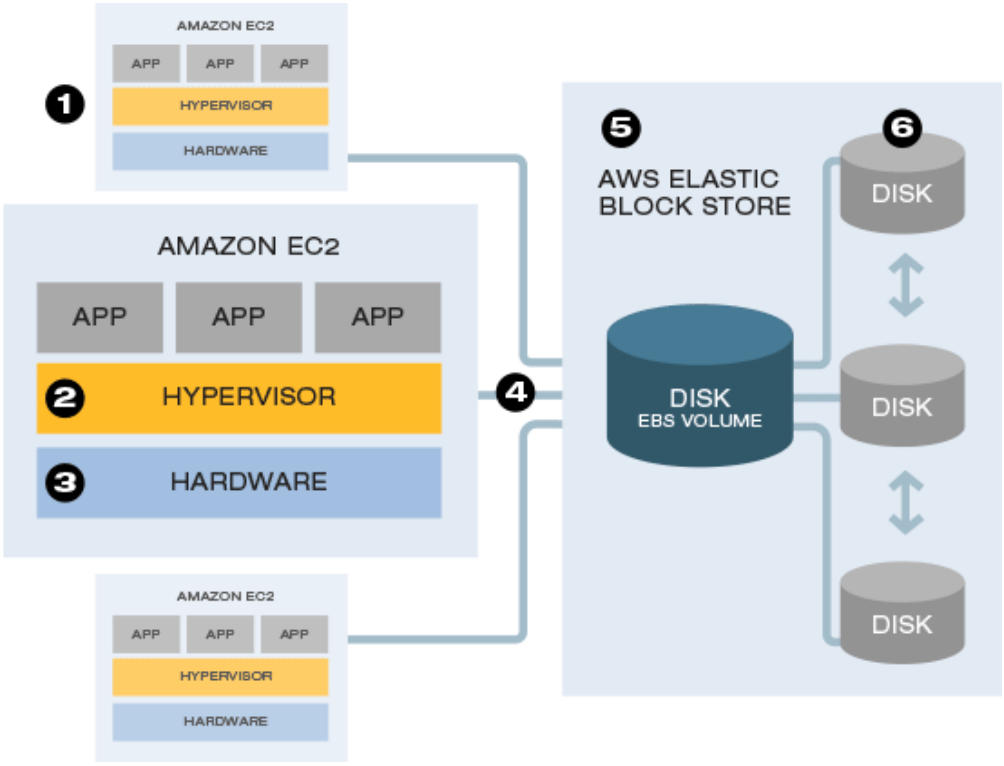

Figure 3.3 - Amazon EC2 architecture [41]

Amazon separates logically disk from the rest of the hardware, all EC2 servers are network-attached to Elastic Block Store (EBS) disks. This allows to isolate problems and tune performance in either parts. As for the hardware and the hypervisor, it is not disclosed but the hypervisor relies on Xen.

3.3.2 Eucalyptus

Eucalyptus is an open-source IaaS application suite, developed with a close partnership with Amazon. Once deployed on production, the resource pool can dynamically scale up or down depending on application workload demands. The agreement with Amazon Web Services allows to maintain fidelity on API compatibility and therefore deliver hybrid Cloud capability between AWS and Eucalyptus environments [42]. Eucalyptus supports Xen, KVM and ESX hypervisors. Its architecture is demonstrated in Figure 3.4.

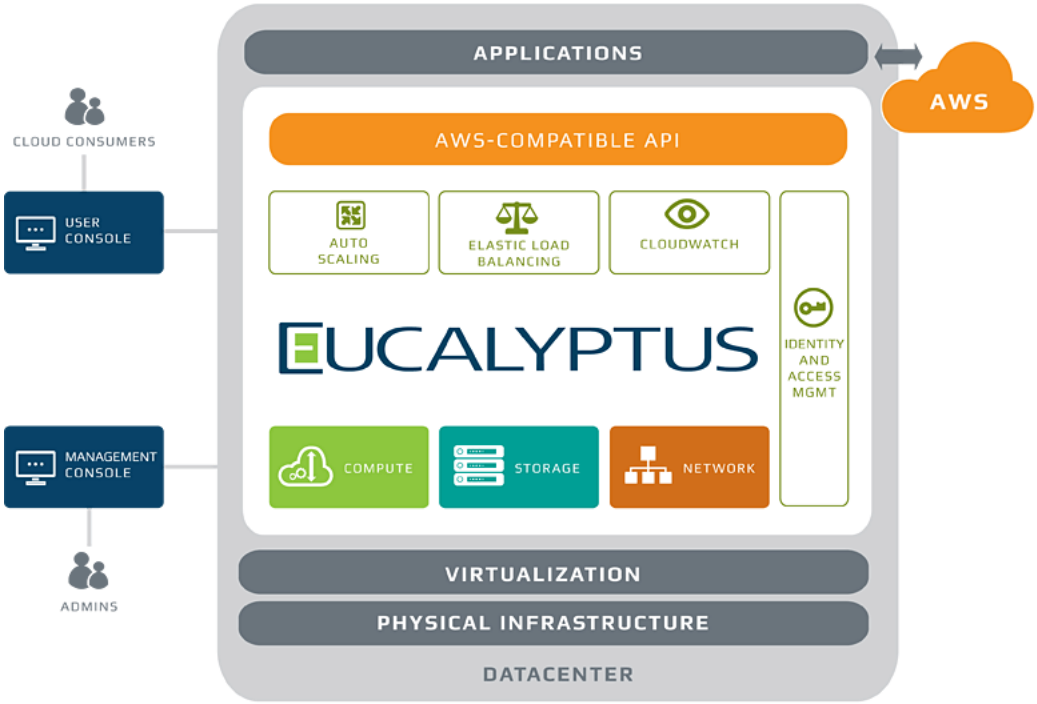

Figure 3.4 - Eucalyptus architecture [42]

Above Eucalyptus' virtualization layer, three functional blocks co-exist: compute, storage and network. All remaining blocks work around these, including the API to connect to AWS.

3.3.3 OpenStack

OpenStack is also an IaaS developed along Amazon EC2 standards. The project emerged in 2010 when Rackspace, allied to NASA, launched a first edition to fulfill its different philosophy about Cloud architecture and open-source software. Its rapid growth amongst the community has been an indicator of great acceptance.

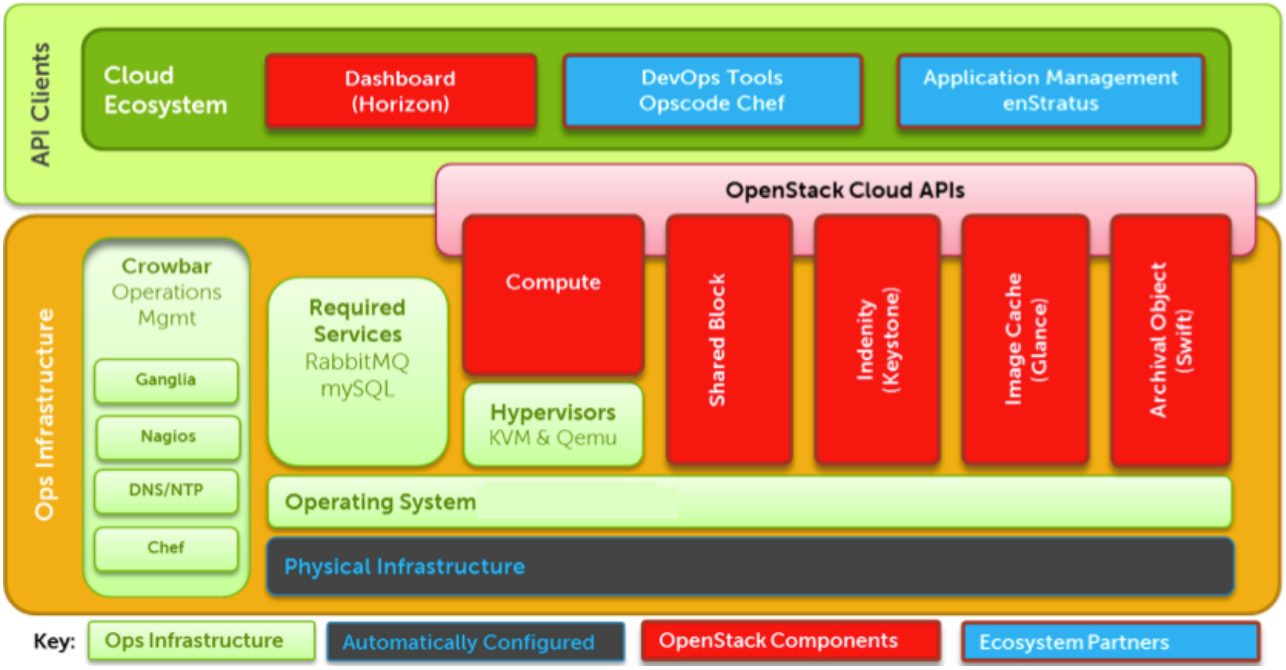

Figure 3.5 - OpenStack architecture [18]

The Openstack IaaS incepts different models in different components. It has several components, with relevance to Nova (computation), Swift (storage), Neutron (networking), Keystone (identity), Glance (image) and Horizon (managing dashboard). Its architecture is depicted on Figure 3.5. It

supports KVM, XEN and VMWare hypervisors. To be noted the particular emphasis on modular components, which are developed independently thus causing less problems while having continuous updates.

3.3.4 OpenNebula

OpenNebula is an open-source on-premise IaaS, offering a flexible solution for the comprehensive management of virtualized data centers [43]. It appeared in 2008 with a first release, developed to fulfill private Cloud needs by the Universidad Complutense de Madrid. It has grown to be one of the most widely used private IaaS and its developing community and continuous updates persist its success.

OpenNebula is written in Ruby, Java and Python, providing a platform-independent software. It has a modular and extensible architecture with customizable plug-ins over the main core, which manages virtualization, images, storage, network and monitoring, among others. Its architecture is illustrated on Figure 3.6. It supports KVM, XEN, VMWare hypervisors and Hyper-V.

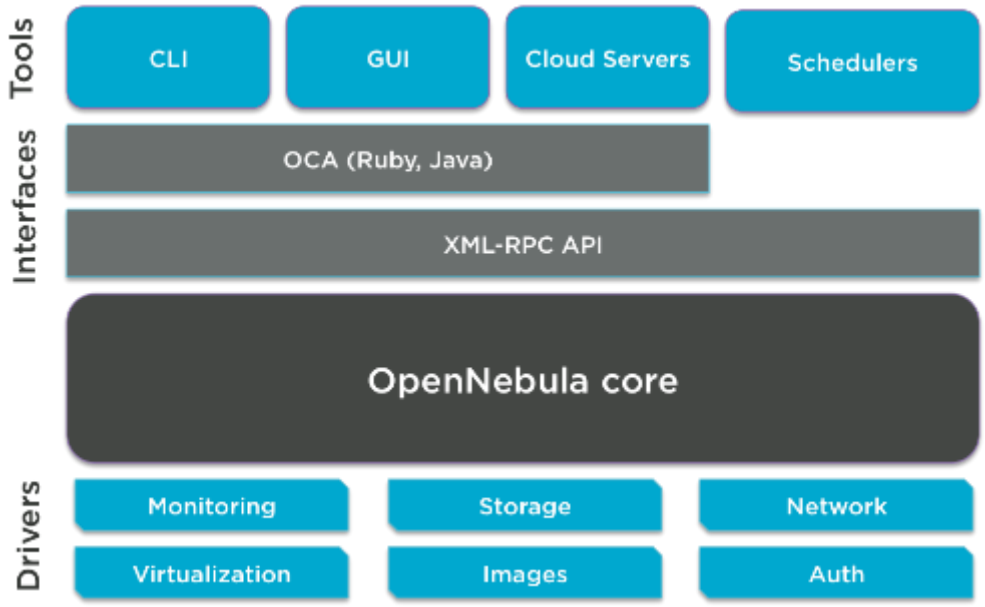

Figure 3.6 - OpenNebula architecture [43]

OpenNebula sits on top of the installed OS hypervisor, using small executables (drivers) to deliver functionalities. The modular capability is relevant, as it allows modifying and replacing small parts by customized ones. The presented interfaces, with REST APIs for OCCI and EC2, but also in Java and Ruby are also relevant to extend its features.

3.3.5 OpenQRM

OpenQRM is an open-source IaaS for managing heterogeneous data center infrastructures, allowing building of private, public and hybrid IaaS. It started in 2004 as an initiative of Qlusters, passed to the community and now it is managed by OpenQRM Enterprise [44].
This IaaS is accessible by either a web Interface, the HTTP API or the command-line and combines different resources and technologies with plugins developed in C. It supports KVM, XEN, VMWare hypervisors and also OracleVM. Its architecture is shown in Figure 3.7.

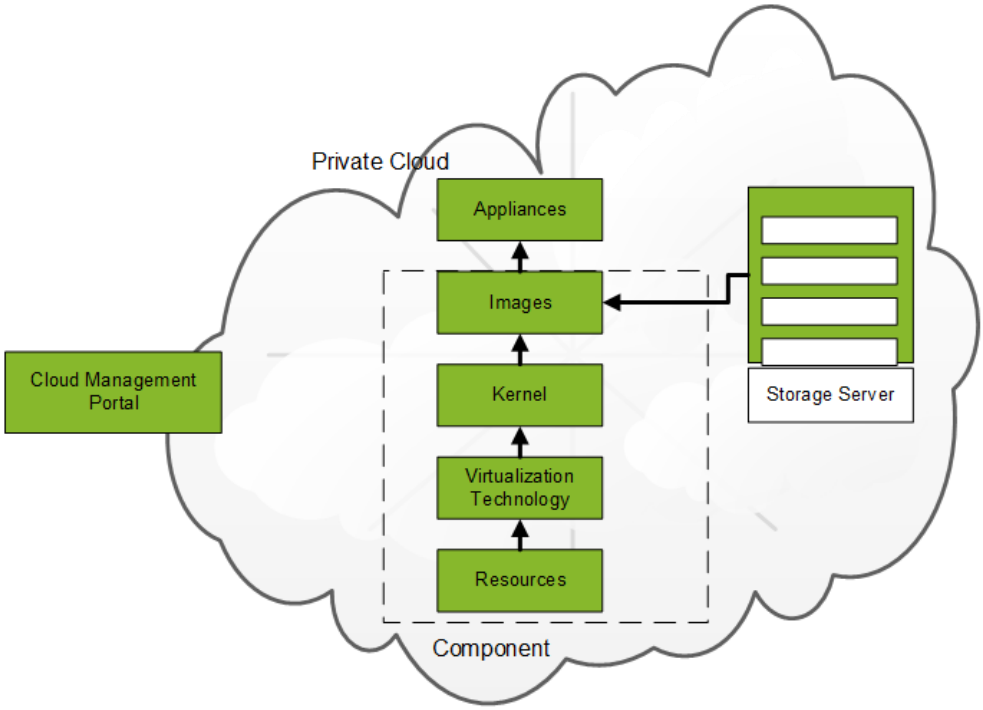

Figure 3.7 - OpenQRM architecture [44]

Sitting on top of the hypervisor, OpenQRM uses a specific kernel, customizable and replaceable, to deploy images. The storage block is logically separated, even though is can be used as a local disk. The web portal is able to manage the IaaS, except for the storage server.

3.3.6 XenServer

XenServer is a native 64-bit IaaS, designed to assure the scalability required by business-critical applications, developed by Citrix. It combines the highest host and guest CPU/memory limits available, coupled with resource controls for CPU, network and disk [20]. Citrix XenServer is an enterprise-ready, Cloud virtualization platform containing capabilities to create and manage a virtual infrastructure. It is built over the Xen hypervisor with an agent which makes remote calls and manages communications [4], delivering both Windows and Linux servers. The IaaS starts with free version and offers also a premium edition. Figure 3.8 shows XenServer architecture.

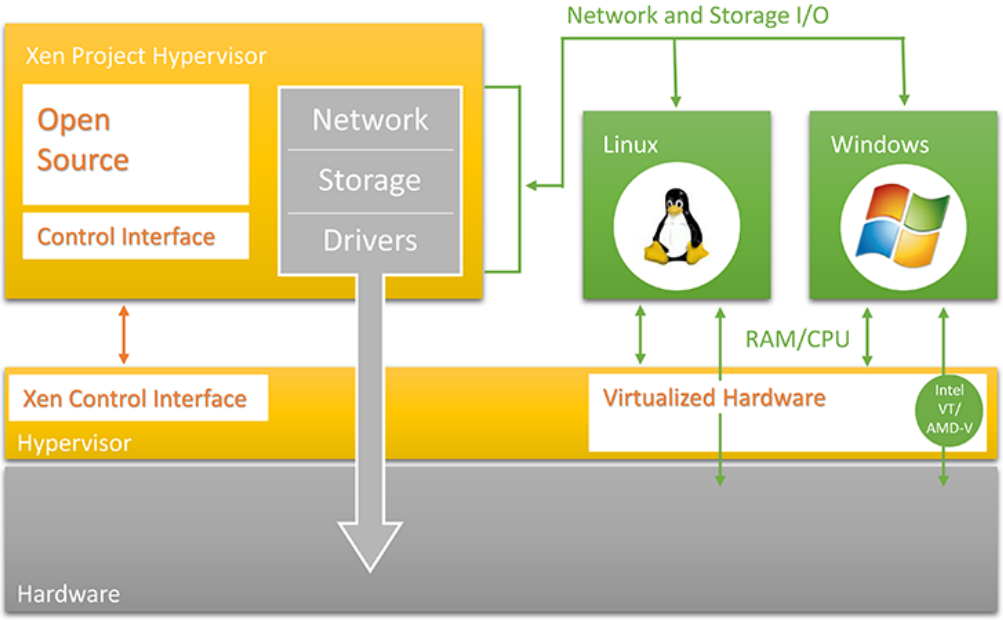

Figure 3.8 - XenServer architecture [20]

With a type 1 hypervisor, XenServer uses dom0, the main domain with OS with special permissions, to control network, storage and drivers. This domain allows to provide the virtualized hardware to deliver VMs.

3.3.7 Oracle VM

Oracle VM is an IaaS provider that features an environment for better leveraging the benefits of virtualization technology [45]. Oracle VM uses the Xen hypervisor to virtualize resources and divides in two functional elements: Oracle VM Manager and Oracle VM Server. The VM Manager provides a web interface to manage VM lifecycle, resources, templates and disks. On the other hand, the VM Server presents a self-contained virtualization environment to allow running VM and includes a Linux kernel with support for a broad array of devices, file systems, and software RAID volume management. Figure 3.9 depicts its architecture.

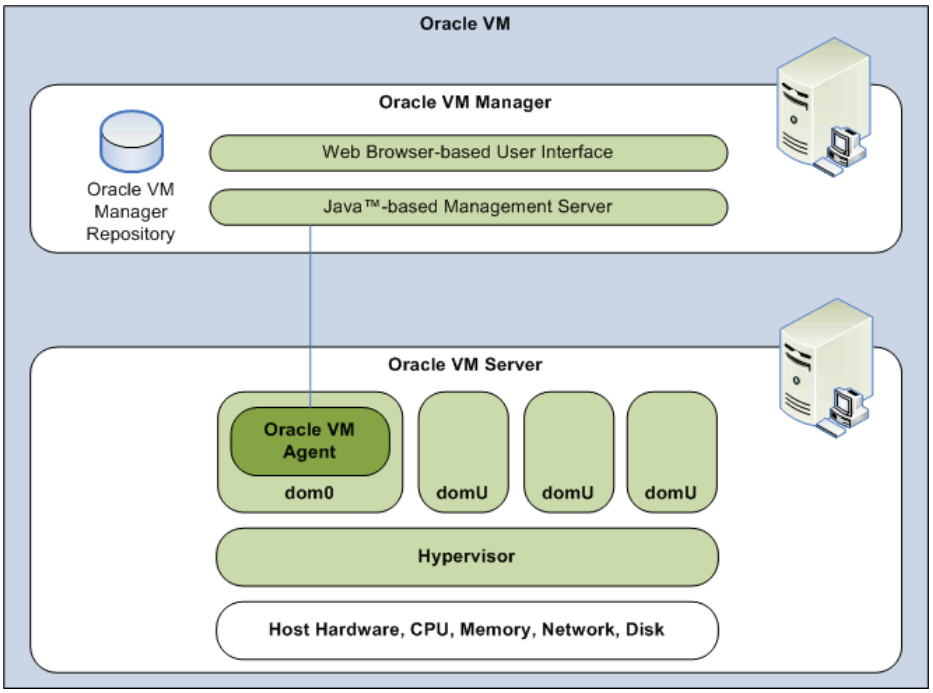

Figure 3.9 - Oracle VM architecture [46]

Much alike XenServer, Oracle VM has a privileged dom0 guest OS, which is used to control the hypervisor and the resources. This OS has a VM Manager running, which makes the connection between the user and the infrastructure.

3.3.8 CloudStack

Cloudstack is an open source software for creating, managing, and deploying infrastructure Cloud services [47]. It emerged on the market as a late player in 2010 by Cloud.com. Quite similar to OpenQRM, CloudStack is developed in Java and uses its framework with a plugin system architecture, providing support for virtualization technology through them. Its main core controls computing, network and storage seamlessly and delivers them to the above layer, the so-called orchestration engine, as shown on its architecture in Figure 3.10. Cloudstack supports KVM, XEN and VMWare hypervisors.

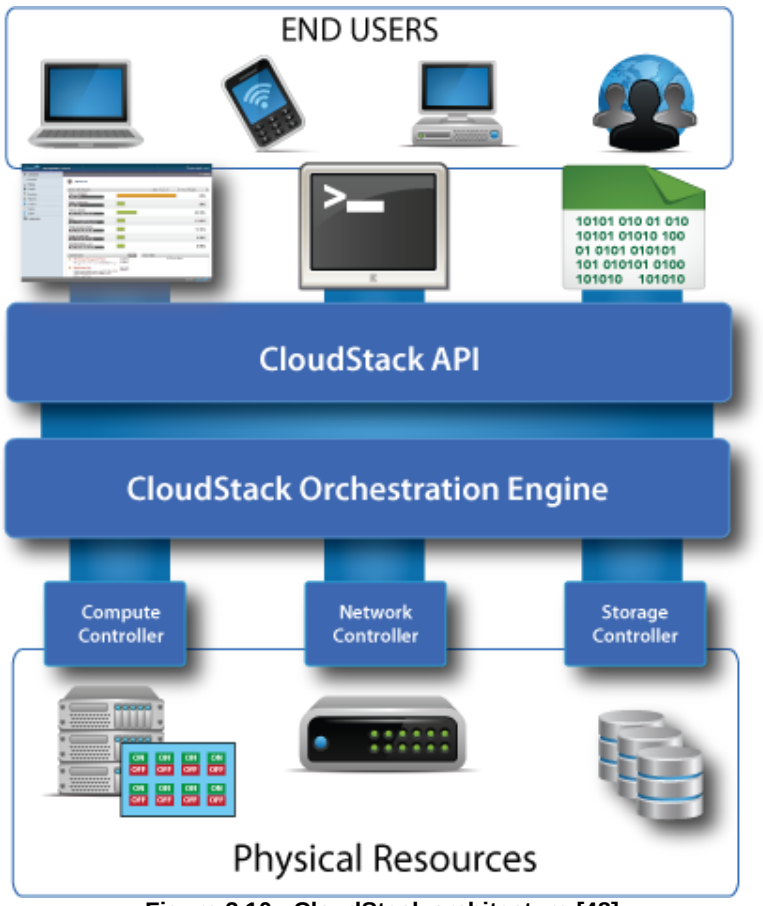

Figure 3.10 - CloudStack architecture [48]

3.3.9 Comparing IaaS

There are several conceptual differences between all applications, but all have command-line interface, web interface, image and storage management, and additional control over the API. There are many variations, like the programming language, some use Java and C, others Ruby and Python. But they show distinct adaptability to the supported hypervisor, which can be realized in Table 3.1.

Table 3.1 - IaaS Provider comparison on supported virtualization

Provider	Xen	KVM	VMWare	Hyper-V
Amazon EC2	X			
Eucalyptus	X	X	X	
OpenStack	X	X	X	X
OpenNebula	X	X	X	X
OpenQRM	X	X	X	
XenServer	X			
Oracle VM			X	
CloudStack	X	X	X	

Open-source IaaS have the capability of more than one hypervisor, but that is inherent from the business model. About open-source IaaS and beyond software capabilities, there is also its supporting community, which is somehow an indicator of continuous development.

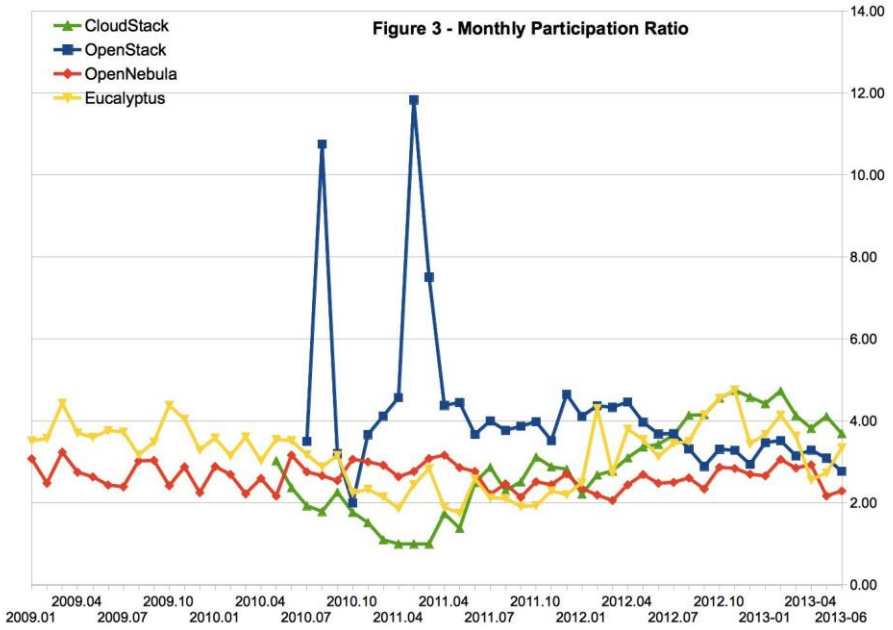

Figure 3.11 - Open-source IaaS monthly participation ratio [49]

On this data, all IaaS, with a few less participation from OpenNebula, are very tied-up, though the average remains higher for OpenStack. On this comparative, there is also more data being compared, including the total number of topics (threads), messages (posts), and participants (unique email addresses or registered members) [49].

3.4 Cloud Features

There are several characteristics that a Cloud should have to be considered one, but there are five stated as fundamental on NIST definition: on-demand self-service, rapid elasticity, resource pooling, broad network access and measured service. However, and since the Cloud is in an advanced maturing phase we can additionally point out virtualization and advanced security as an indispensable set of features.

On-demand self-service

One of the most important features of the Cloud that has allowed its massive growth is the capability of users to provision computing resources by the amount they want and when they want, automatically. Normally this is made from a web page, terminal or API. The infrastructure processes the request, determines resource location, reserves resources and returns privileged access to the user.

Rapid elasticity

One of the Cloud's most powerful inner capabilities is its agility to quickly scale services and applications up or down. Resources can automatically be allocated or released according to user or application needs. Another extremely useful point is that the existing abstraction layer gives a perception of endless resources to the user, which can be provisioned at any time.

Resource pooling

The Cloud multi-tenancy model allows the resources to be pooled, serving multiple users with diverse resources, allocated and reallocated on-the-fly. This applies to resources like CPU, storage, memory and also network. The management layers permit having pools with resources in different locations, but acting like a single place.

Broad network access

Another excellent feature in the Cloud is ubiquitous access; it can be accessed and managed from clients ranging from a powerful workstation to a smartphone. One can either choose to use SSH, HTTP or another authorized and configured protocol.

Measured service

With a sustained growth as a business model, the Cloud has included monitoring since its beginning. Resources, independently of their location, are controlled and measured to provide accurate management and billing. This gives a good sense of reliability and helps on the Cloud agility.

3.5 Cloud Problems

As any technology, Cloud Computing also has many issues. Some are tech-related, but also there are privacy, security and even environmental issues.

Privacy issues

One of the main advantages of Cloud Computing is data portability. Users can save and move data to almost any provider and location. But while talking about the Cloud, the whole word/concept seems to abstract a lot more than we realize. This characteristic has proven itself as a technology's Achilles heel.

Whatever the data type or the business category, every personal computer has private data that should remain confidential. When changing the paradigm to the Cloud, we lose control of our own data. Where is our data? Who can access it? If it is abroad, what laws apply and how do they conflict with national and other international laws? This conflict between domestic and international data privacy laws arises a new barrier to break.

In the US, the legislation created after the 11th September allows unwarranted search and seizure of any data on US firms. As the leading Cloud supplying country [50], that makes any customer uncertain of his privacy. Some countries have specific rules in some areas about data storage. For instance, Portugal health records must be contained in domestic territory and only available to medics or assessed patients; some Canada provinces restricted public services data storage and access to Canada [51]. Lately there are major advances: Amazon supports compliance with most of the US policies & certifications and the European Comission has released several references in the last version of the directive for the processing of personal data [52].

Security

One of the top concerns about the Cloud is its level of security. All that layers of abstraction create the perception of losing control of all our data, and its effect increases with distant resources, when the Cloud is public. But looking to the problem as a whole, we can state that is no less secure than any other data traveling around the network systems. There are traditional solutions like VPN, IPSec, HTTPS and many more, in addition to firewalls and storage data encryption. VM deployments are configurable, network access is narrow and by default only a few number of ports are open and initialized.

Yet, some problems do not really apply when using a Cloud logic with HPC services: VMs are created and discarded after job completion, even if they are attacked the threat can be contained in PaaS level and is soon to be cleaned-up. The resulting data can be more sensitive because it is returned

by the network. However, this is clearly an important asset when comparing to traditional datacenters. For example, the DHCP service, provided by QEMU, issues a VLAN IP to both host and VM, enabling communication between both machines without outside interference. This modeling is extremely safe and useful, because when deploying VMs no outside probes are alerted, and therefore cannot hack it.

Still, Cloud attacks exist in several forms: denial of service, malware injection, metadata spoofing, XML signature wrapping, VM-to-hypervisor attack and Cross-VM side-channel attack [53]. One of the most dangerous is the hypervisor attack, which exploits the hypervisor's vulnerabilities to gain access to the host machine, allowing the chance to gain administrative rights and compromise service.

Memory limitation

Virtual memory ballooning is an approach used by a hypervisor to allow the physical host system to borrow unused memory from certain VMs and share it with others [19]. Memory ballooning allows overcommitting the total amount of Random Access Memory (RAM), exceeding the amount of physical RAM available on the host. When the host system runs low on physical RAM resources, the ballooning process allocates it selectively to VMs, removing memory from some guests to give temporarily to others.

Despite this method to share memory within a host and the innumerous Cloud capabilities, there is a hardware limitation on RAM when it comes to share memory between hosts. This is a technological limitation, because RAM is an embedded resource which makes harder for the hypervisor to share [54]. One of the biggest problems is that a shared memory view does not provide any distinction between local and remote data; this can cause performance degradation in a Cloud when individual nodes could possibly be allocated in physically distant nodes [55]. Some programs require large amounts of RAM, conservatively about 100GB or more, and the largest node around the several Clouds is only 32GB.

Applications consuming large amounts of RAM are likely unfeasible on the Cloud because they need large amounts of close memory and the memory across the Cloud resources cannot be shared efficiently as a whole. For instance, genomic/transcriptomic assembly programs have very large sequences of data that cannot be processed separately – the next results depend on the last processed results and in no specific order. Unlike VMs disk or CPU, there is no easy way to share RAM across hosts. However, there are projects like RAMCloud [56] and Hecatonchire [54] (educational) or Terracotta [57] and Gemfire [58] (commercial) that have currently working trends. It totally depends on the nature of the application.

Using approaches as Hadoop [59], MPI [60], MOSIX [61] or VSMP may not solve the problem, because these technologies are helpful when you could partition your application into concurrent executing blocks.

If the application is partitionable into parallel/concurrent blocks, we choose the best technology that fits our needs. Otherwise, there are almost no options other than to upgrade the hardware. When choosing the technology for the application, we must take in consideration:

- If the application is process intensive and there are many passing messages, use MPI or OpenMP [62];
- If the application is data intensive, use Dryad [63] or Hadoop;
- If it contains many light-weight threads and matrixes, use GPGPUs.

Environment

One of the advantages of using the Cloud is the datacenter maintenance, mitigating both human handling and costs as power and cooling. However, the competition to build and deliver the Cloud

leaves some aspects aside. How much energy is required to power the ever-expanding online Cloud? What share of global greenhouse gas emissions belongs to the IT sector?

A recent research of Greenpeace [64] has gathered information about the major datacenters' locations and specifications, estimates Cloud datacenters to consume an equivalent to 180 000 homes. The study, divided into 4 categories - Energy Transparency, Infrastructure Siting, Energy Efficiency & GHG Mitigation, and Renewables & Advocacy - indicates that Amazon, Apple and Microsoft have unsustainable practices, such as engaging in rapid growth without an adequate energy strategy and for powering their clouds with non-green energy. Figure 3.12 shows Greenpeace scorecard, referring well-known providers. A to F are green ranks, A stands for the best practices and F for the worse.

Company	Clean Energy Index	Coal	Nuclear	Energy Transparency	Infrastructure Siting	Energy Efficiency & GHG Mitigation	Renewables & Advocacy
Akamai	NA	NA		A	C	B	D
amazon.com web services	13.5%	33.9%	29.9%	F	F	D	F
	15.3%	55.1%	27.8%	D	F	D	D
DELL	56.3%	20.1%	6.4%	C	C	C	D
facebook	36.4%	39.4%	13.2%	D	B	B	C
Google	39.4%	28.7%	15.3%	B	C	B	A
hp	19.4%	49.7%	14.1%	C	D	B	C
IBM	12.1%	49.5%	11.5%	C	D	C	D
Microsoft	13.9%	39.3%	26%	C	D	C	C
ORACLE	7.1%	48.7%	17.2%	D	D	C	D
rackspace HOSTING	23.6%	31.6%	22.3%	C	C	C	C
salesforce	4%	33.9%	31%	B	C	C	C
twitter	21.3%	35.6%	12.8%	F	D	F	D
YAHOO!	56.4%	20.3%	14.6%	C	B	B	B

Figure 3.12 - Greenpeace Clean Energy Scorecard

On the other side, Akamai and Google have a good planning and execution strategy. Far from being a highly ranked requirement when it comes to choosing a Cloud provider, the green rank can be at least a catalyzer for the coming Cloud development. By demanding more from providers, Cloud companies can go towards the development of cleaner electricity generation that will ensure a better environment.

Storage limitation

Regardless of all attempts to smooth performance limitations there are no miracles on this chapter. Intensive I/O appliances like some applications that rely on heavy databases do not scale well because they have small read/write operations that become impracticable with a wide system with such a variable latency. The number of data blocks constituting the table data has a direct influence

on disk I/O as the more the number of data blocks the higher the disk I/O thus affecting the overall system performance [65]. The impact factor on performance depends on the application model. Lately this issue has been mitigated with the introduction of Solid State Drives (SSD) on the Cloud infrastructures, either used as disk cache or replacing regular storage.

3.6 Cloud Standardization

The emerging of the Cloud era was a globally disperse evolution, scattered by different hardware and technologies but also in infrastructure and composing layers. Early attempts for standardization were rather shy because Cloud computing originated in the private sector [66] . As a scattered development, with few defined standards, each provider developed its Cloud infrastructure on their own, instead of using normalized settings, a very similar situation to the appearance of the TCP. This vision from within held back a bit on standardization [67].

Standards are important because they ensure interoperability in a well-designed formalism [68]. They define a persistent basis to allow sustained development, even though they can be a bigger start up obstacle. Thus, developing under standards can be more costly and harder at the beginning of the process, but in the long run it compensates in benefits [68]. However, it is widely accepted that common best practices and standards will be needed to realize many of the benefits being applied for Cloud Computing [69].

Nonetheless, Cloud Computing usage went sky high because of its major functionalities: on-demand use, ubiquitous access, pay-as-you-go service, resource saving and elasticity. As an obvious result, the convergence of the whole Cloud eco-system turns into a colossal task, restricted in edgeless connectors and in the business lobbies that cause a dreadful inertia.
Interchangeability, more than interoperability, is the path to follow. By having a common representation for operationally important artifacts, practitioners have some flexibility to move models from one environment to another [68]. There are some proposals of evaluation techniques to determine the best Cloud service [70], but is avoidance the way? We change Internet and TV provider as we want, without changing laptops or TV sets. Why shouldn't applications be moved across providers when we decide to change the service supplier? Usually public providers defend their business model with technology lock-ins, thus making portability more difficult in order to avoid users to easily switch provider.

No doubt that with standardization Cloud Computing would evolve to its maximum potential and would give users full throttle to their applications. Fulfilling the cross-Cloud paradigm is a hard assignment but it surely would free the users to the Cloud sky. In [71], the author describes the value chain of standards in a clearer way, as shown in Figure 3.13; broad markets tend to increase competition, which leverages technology developments.

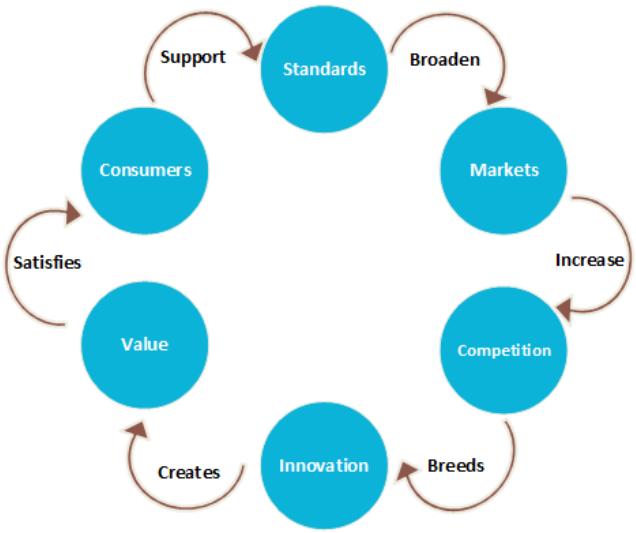

Figure 3.13 - Standards value chain [71]

This intrinsic property that makes the process circular benefits not only the providers, with more consumption, but also the users that will get everyday more enhanced services.

To address these communication problems between clouds, several organizations joined in combined efforts to deliver Cloud standards. The European Telecommunications Standards Institute (ETSI) has started a project for standardizing the use of Grid and Cloud computing technology in the context of telecommunications, with the formation of the Technical Committee GRID [72]. These include analysis of grid and Cloud interoperability gaps and surveys, comparisons between Grid, Cloud and telecommunication systems among others related to grid and Next Generation Networks.

The IEEE society also created a study group in 2009 for Cloud standards, with a call for participation open to all interested. The IEEE Standards Association believes that a major impediment to the growth of Cloud computing is the lack of comprehensive high-level portability and interoperability standards [73]. The IEEE Cloud Computing Initiative has originated two working drafts, which express the position of experts from 160 countries [74].

The Distributed Management Task Force (DMTF) also assumed the need for open management standards for Cloud Computing in 2009 [75]. DMTF focused its goal in developing Cloud resource management protocols, packaging formats and security mechanisms to facilitate interoperability and portability between compute clouds. NIST and the International Telecommunication Union (ITU) also initiated programs to study Cloud Computing's standardization ecosystem [74].

Already set is the open standard for Cloud images. The Open Virtualization Format (OVF) was released by DMTF in 2009 and updated to version 2.0.1 in August 2013 [76]. It defines a portable and extensible format for software to be run in virtual machines. OVF enables simplified and error-free deployment of virtual appliances across multiple virtualization platforms with a common packaging format for independent software vendors enabling cross-platform portability. The problem is that there are not many providers supporting the OVF format; currently only Oracle, Red Hat, VMware and IBM support this format type along with some OS such as SUSE Linux [77].

A widely used standard for inter-communication is the Open Cloud Computing Interface (OCCI), which is one of the most used protocols for accessing the Cloud providers. It provides a boundary protocol and API that acts as a service front-end to a provider's internal management framework.
Also in a phase of maturation stand the protocols CIMI (Cloud Infrastructure Management Interface), for infrastructure, CDMI (Cloud Data Management Interface), for data, and TOSCA (Topology and Orchestration Specification for Cloud Applications) for topology [78].

Chapter 4

4 High-Performance Computing

HPC is a particular area which aims to solve complex scientific problems and process data in a fast and efficient way through intensive computing. It has become increasingly popular since the late 90's, although it has begun in the decade of 50 [79]. Supercomputers have transformed a wide number of science and engineering fields, leading to a paradigm shift. With the hardware evolution (new architectures, multicore CPUs) combined with parallel computing, its inherent attribute of dramatically shortening computing times has placed this field on the highest priority, both for researchers and business corporations.
Nowadays, scientists and researchers can simulate large-scale models and applications in a shorter timeframe. With such tools, some theories can be proved/discarded along with their observations in a reasonable time. Figure 4.1 shows a graphic of computing capacity that began in 1993 when Top500 Supercomputers first began collecting the information [80], and also prediction of future capacity. The higher line is the sum of capacity, the middle line is the fastest computer and the lower line is the slowest supercomputer on the Top500.org.

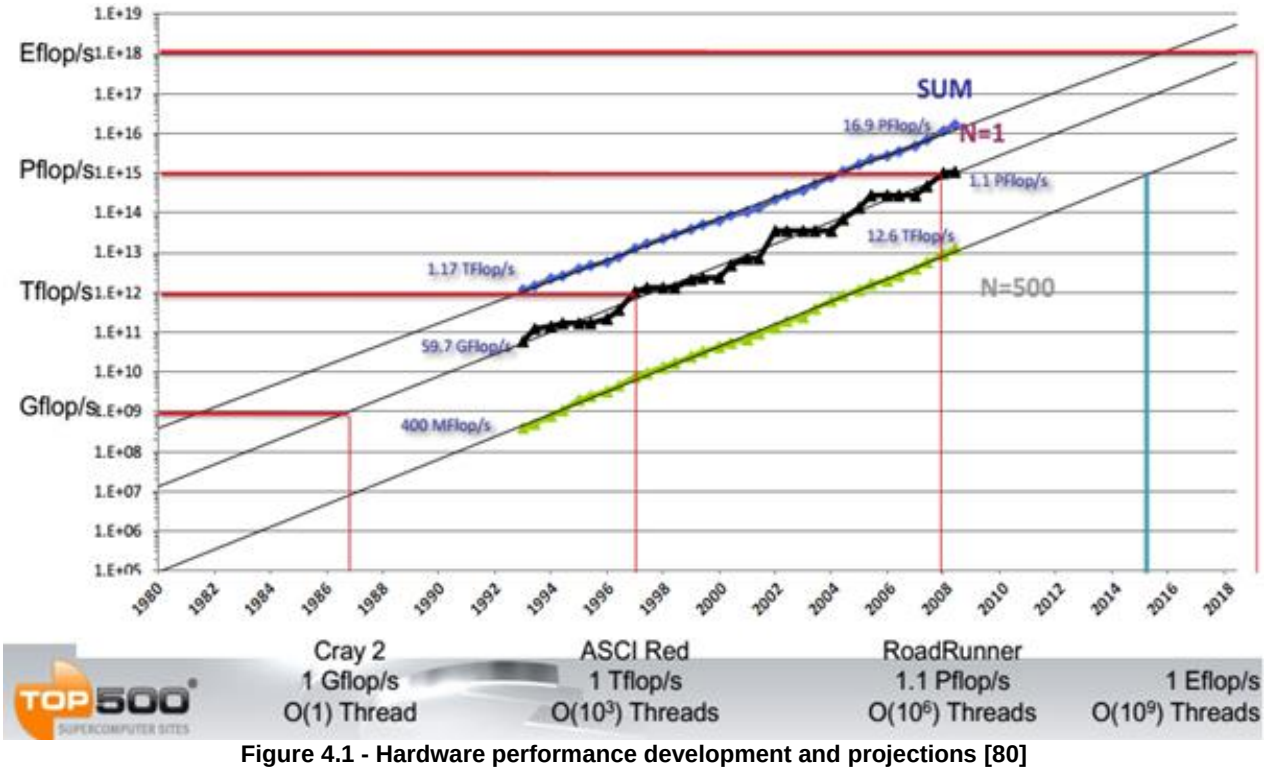

Figure 4.1 - Hardware performance development and projections [80]

In this area, scalability and load balance are major factors when trying to achieve high performance on parallel machines. Typically, an application can be scalable if larger parallel implementations can

solve larger problems proportionally in the same running time as smaller problems on smaller implementations. A load balance generally means all processors have the approximate amount of jobs, to avoid waiting for a processor to complete the solution. This can be challenging in problems which do not have an estimated completion time until they are computed.

HPC and High-Throughput Computing (HTC) are often confused. Although there is no single definition for both, HPC systems allow users to run a single instance of parallel software over many processors. On the other hand, HTC systems usually enable users to run multiple independent instances of software on multiple processors simultaneously. HPC typically delivers large numbers of computing resources over a short period of time to accomplish computational tasks; HTC can deliver large amounts of processing capacity over a long period of time and does not imply the use of state-of-the-art hardware [81].

Both HTC and HPC are designed to be scalable, and can achieve massive computing power by using hundreds of individual processors (cores) to process their tasks.

4.1 Applications

HPC is a comprehensive term that represents several compute-intensive applications that need a large computational power. Applications range from bioscience, medical research, oil exploitation, to financial trading, data warehousing and many more. Table 4.1 shows some fields using HPC and their applications, widely spread over innumerous areas.

Table 4.1 - Applications that benefit from HPC/HTC

Area	Application
Medical	CT scan (pre-processing, reconstruction) 3D ultrasound real-time X-ray
Financial	Derivatives trading Black Scholes model BGM/LIBOR market model Monte Carlo simulations
Oil and Gas	3D imaging processing Reservoir modeling Seismic data interpretation
Weather	Weather forecast Environment evolution study
Movie industry	Computer graphic animated films (Shrek, Madagascar, etc.) Scenario rendering on movies
Biology	Gene and protein annotation Drug therapy mapping to individual's genes Molecular dynamics Pharmaceutical research
Car industry	Crash test simulations Aerodynamics modeling
Other	Data warehousing, military applications, data compression, coder/decoder, search engines, security analysis

4.2 Enabling technologies

There are two main strategies concerning parallel problem solving: task parallelism and data parallelism. Task parallelism is the concurrent execution of many different functions on multiple cores, with possibility of shared data. Data parallelism is the simultaneous execution of a single function across the elements of a dataset on multiple cores. In data parallelism, the application is decomposed by subdividing the data space over which it operates and assigning different processors to the work associated with different data subspaces. Data parallelism is the most common strategy for scientific programs on parallel machines [79] but one of the hardest as well.

Once a strategy is chosen, a programming model must be selected. The two most common programming models in HPC are the message passing model and the shared-memory model. In the message passing model each processor is assumed to have its own private data space and data must be explicitly moved between spaces as needed; in the shared-memory model, it is assumed that all data structures are allocated in a common space that is accessible from every processor [79]. Implementations of these popular models are described briefly on the following sections.

MPI

MPI is a Message Passing Interface protocol used to program parallel applications. It began to be developed in the early 90's and has become de facto standard library for parallel processing on high-performance computing systems [82].

In MPI, data is distributed across the processor memory; if a processor needs to use data that is not stored locally, the processor that owns that data must explicitly send the data to the processor that needs it. The latter must execute an explicit receive operation, which is synchronized with the send, before it can use the communicated data [79]. Figure 4.2 shows a possible MPI architecture.

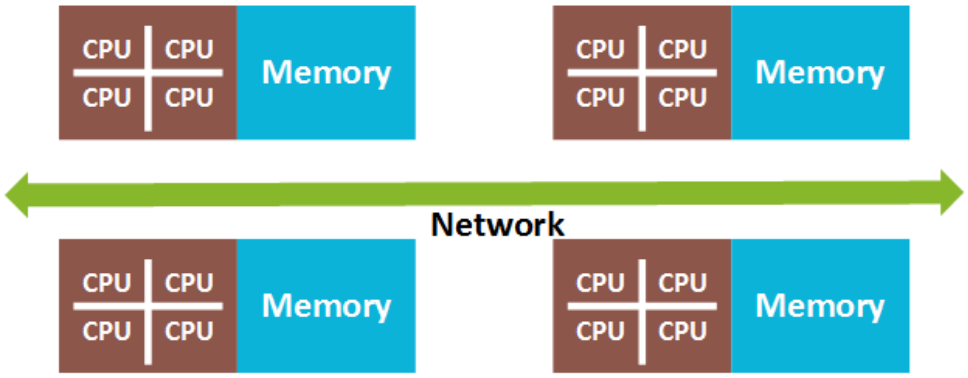

Figure 4.2 - MPI model architecture [60]

MPI 2.0 was launched in 96 but in 2012 an updated 3.0 version of MPI was released, with major features. Now it incorporates a hybrid model that seamlessly handles both types - distributed and shared - of underlying memory architectures [83].

OpenMP

The Open MultiProcessing stands for a multi-platform shared memory multiprocessing API that allows developing parallel applications for a computation on several CPU architectures and operating systems. OpenMP programs accomplish parallelism exclusively through the use of threads. A thread is the smallest unit of processing that can be scheduled by an operating system. The idea of a subroutine that can be scheduled to run autonomously might help explain what a thread is [84]. OpenMP is a multi-threaded, shared memory parallelism, as shown on Figure 4.3. As it is an API, it depends on three major components: compiler directives, runtime library routines and environment variables.

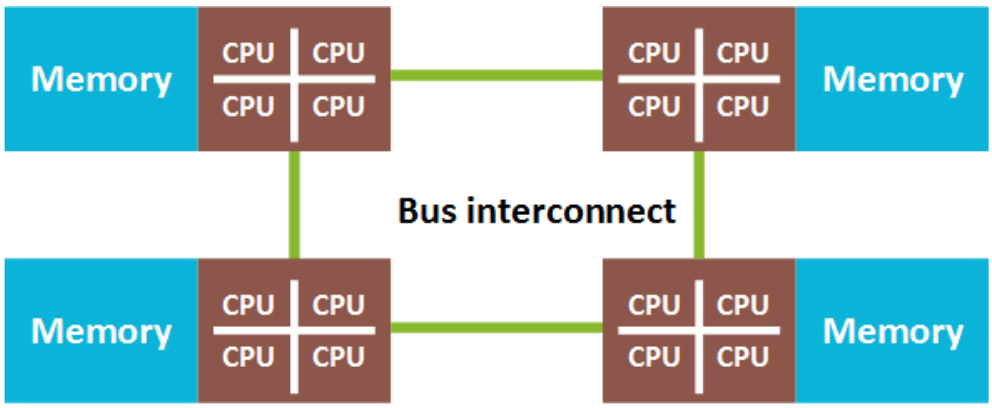

Figure 4.3 - OpenMP model architecture [60]

Emerging on the early 90's, vendors of shared-memory machines supplied similar directive-based, programming extensions in the Fortran language. The programmer would expand the program with directives specifying which loops were to be parallelized and the compiler would be responsible for automatically parallelizing such loops [84]. Only in 1997 was the standard specification released, followed by 2.0, 2.5, 3.0 and 3.1 versions in 2002, 2005, 2008 and 2011 respectively, with extended compatibility with programming languages: after Fortran came C and C++.

MPI versus OpenMP

Although there is a great discussion between both approaches, the fact is that they are rather different and utilization depends on the computing algorithm. There is no better performance comparison when the intrinsic nature of the jobs is very dissimilar, although users can be misled by completion time. For a better perspective, we can detail some advantages and disadvantages of MPI and OpenMP [85]:

Pros of MPI

- runs on either distributed or shared memory architectures
- can be used on a wider scope of applications than OpenMP
- each process has its own local variables
- distributed memory machines are less expensive than large shared memory ones

Cons of MPI

- limited performance to the communication network between the computing nodes
- more difficult to change programming from serial to parallel version
- can be harder to debug

Pros of OpenMP

- can run with serial programming, serial statements usually do not need modification
- easier to program and debug than MPI
- gradual parallelization - directives can be added incrementally
- code is easier to understand

Cons of OpenMP

- can only be run in shared memory computers
- requires a compiler that supports OpenMP
- generally used for loop parallelization

4.3 Alternative approaches

Even though it has been extremely useful and widely used, the MPI programming model just turned 20 years. As the supercomputers prepare for exascale computing, there is a determination to move

beyond MPI. With systems several times larger than today's supercomputers and their ecosystem's heterogeneity, the main impression is that MPI will not deal smoothly with this new reality and will require more engineering from the programmers.

A widely used alternative is MapReduce, a data processing framework for large data sets. Other alternatives being offered are a PGAS (Partitioned Global Address Space) model known as GASPI (Global Address Space Programming Interface) and OpenSHMEM (Open Symmetric Hierarchical Memory).

MapReduce

MapReduce is a distributed computing model and implementation by Google [86]. The model is based on two main functions, map and reduce. On the map function, the main node divides the input data and delivers it to processing nodes, which can repeat the process recursively to other nodes. After computing the minor problem, the output is returned to the main node. On the other hand, the reduce function is performed by the main node, which collects all output data and merges it into the desired solution. Figure 4.4 shows a small example with shapes:

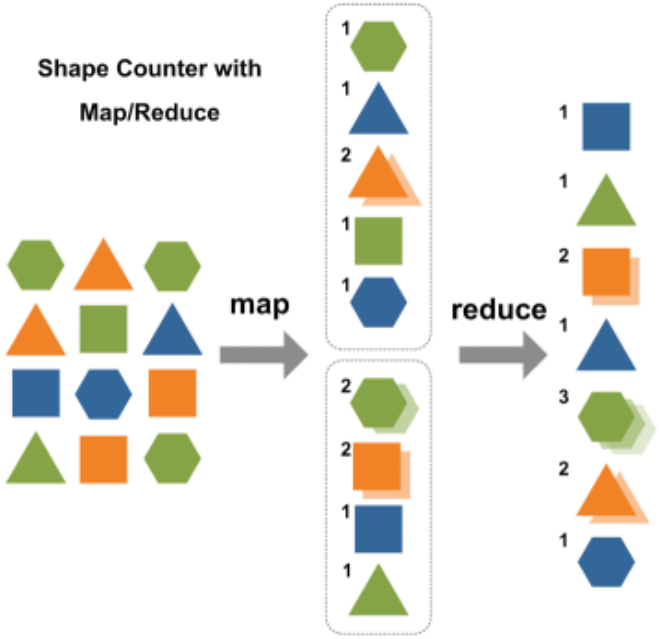

Figure 4.4 - MapReduce model

The inputted map function processes a key/value pair to produce a set of intermediate key/value pairs, and the specified reduce function merges all intermediate values associated with the same intermediate key, in this case, colored shapes.

One of the keys for good performance is its singular filesystem. The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications that have large data sets [86].

GASPI

GASPI is a partitioned space API that centers its development on three main goals: flexibility, scalability and fault tolerance [87]. It provides an API which includes synchronization primitives, synchronous/asynchronous collectives, passive receives, global atomics, communication groups and queues.

One effort being made is that GASPI is fully interoperable with MPI and able to port industry-oriented applications. Being designed to coexist with MPI, it provides the possibility to complement MPI with a partitioned global address space. Figure 4.5 shows the organization of the GASPI address space.

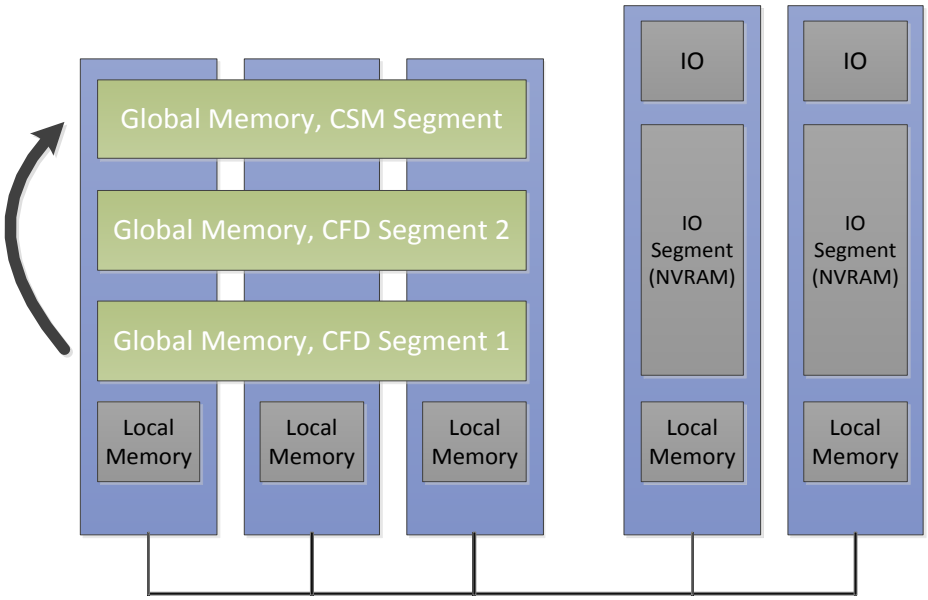

Figure 4.5 - GASPI architecture

While such approach delivers an increased scalability, fault–tolerant execution is not supported due to the limitations of MPI. The project started in June 2011 and finished in 2014, but promising results have been published recently this fall [88] and its extension is being planned.

OpenSHMEM

OpenSHMEM stands for Open Symmetric Hierarchical Memory and it is a standard for Shared Memory library implementations [89]. It consists on a 1-sided PGAS Library developed to run point-to-point and collective routines and allows direct access to remote data without involving the target. Main features include a standardized interface, a shared memory view, portability and an active community to support development. All processors have their own view of symmetric variables, as depicted on Figure 4.6.

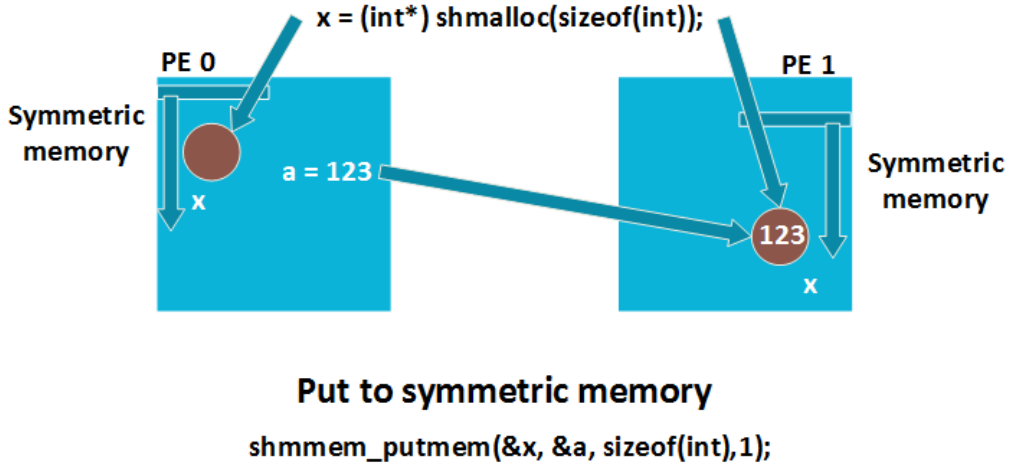

Figure 4.6 - Symmetric allocation

Being a library, there are no requirements for specific compiler or language extensions. For two-sided send-receive operations (MPI), the sender copies data into the shared memory region, and the receiver copies it back. OpenSHMEM eliminates the need for acknowledgement by sharing the data on memory, thus saving half of the signaling messages.

OpenSHMEM is an ongoing project that still needs to mature. Developers aim to build core tests of correctness and performance, as well as to compare implementations of collective algorithms to fulfill both verification and validation criteria.

4.4 HPC in the Cloud

As Cloud Computing is representing a paradigm change in several areas, companies are adopting such paradigm, changing to the pay-per-use model offered instead of building in-house data centers. Several enterprises are deciding to use virtual data centres to facilitate infrastructure management, increase flexibility and trying to decrease the need of hardware maintenance. Data centers are being replaced by dynamic, on-demand and flexible infrastructure that bursts to the Cloud to attach additional compute capacity [1].

On the other hand, assembling a classic data cluster capable of a high throughput still arises as a hard mission. Budget issues, bureaucracy filling, hard-to-estimate needed capacity and a huddle of hardware/software installation which could last for weeks. HPC in the Cloud presents itself as an extremely balanced solution, with fast instantiation, reasonable computing capacity and a pay-per-use billing adequate to peak usage.

4.4.1 Benefits and limitations

Although a Cloud environment induces virtualization, HPC can benefit from it in several ways:

- Elasticity and dynamic resource management, including live migration;
- Heterogeneous environments, any environment/application can be customized to clone on VMs;
- Workload isolation, needed for failure protection and secure multi-tenancy;
- Unification of IT infrastructure, users can run any kind of application on top of it;
- Great benefit from the economies of scale on the Cloud platform, high capacity in minutes;
- Dynamic fault tolerance, done with replication and checkpoints.

One of the advantageous technical inherent to the Cloud is wait time. Prior results in high-end traditional HPC clusters are superior to the Cloud in raw performance. For instance, wait time which is critical and very common on HPC clusters can be a valuable asset on the Cloud. In order to decide between HPC clusters and the Cloud, it is necessary for the HPC systems to provide wait queue data (both historical and current) [10]. This would allow analysis to determine the expected wait time on the HPC clusters, which is clearly a critical factor in which cluster to choose. Abstracting complexity implies additional costs. In the Cloud we get overhead for each resource allocated and there is also the interconnect network speed bottlenecks. But industry is evolving, and allied to new technologies, having a good hardware utilization and resource aggregation can make a powerful HPC infrastructure in the Cloud.

However, in spite of all these advantages, HPC in the Cloud still drives through a sinuous path as it searches to address technical but also cultural barriers. Most HPC solutions continue to be held in the traditional way.

There are further reasons underlying the slow rate of acceptance. The biggest issue is definitely I/O, both storage and network. Many Clouds have slow inner and outer network rates, which can be simple but huge bottlenecks for data-intensive applications. Another Cloud-related issue is data transfer: every GB that enters or leaves the public Cloud is accounted and charged. This can be very expensive, depending on the application. While an increasing dependence on co-processors stands its own set of challenges for HPC scaling, virtualization adds a layer of complexity that can be a dead weight for latency-sensitive HPC workloads.

4.4.2 Current players

Looking at the publicly available data, several companies offer all three types of Clouds, private, public and hybrid, and provide support for both physical and virtual resources. They offer solutions to run most applications on the Cloud but also HPC applications. However, there are no public benchmark results available which can prove these solutions are able to deliver the performance and the scalability needed for HPC.

Gartner's respected technology summary covers all the common use cases for Cloud IaaS, including development and testing, production environments including HPC [90]. Figure 4.7 shows Gartner's Magic Quadrant with major Cloud providers on 2014.

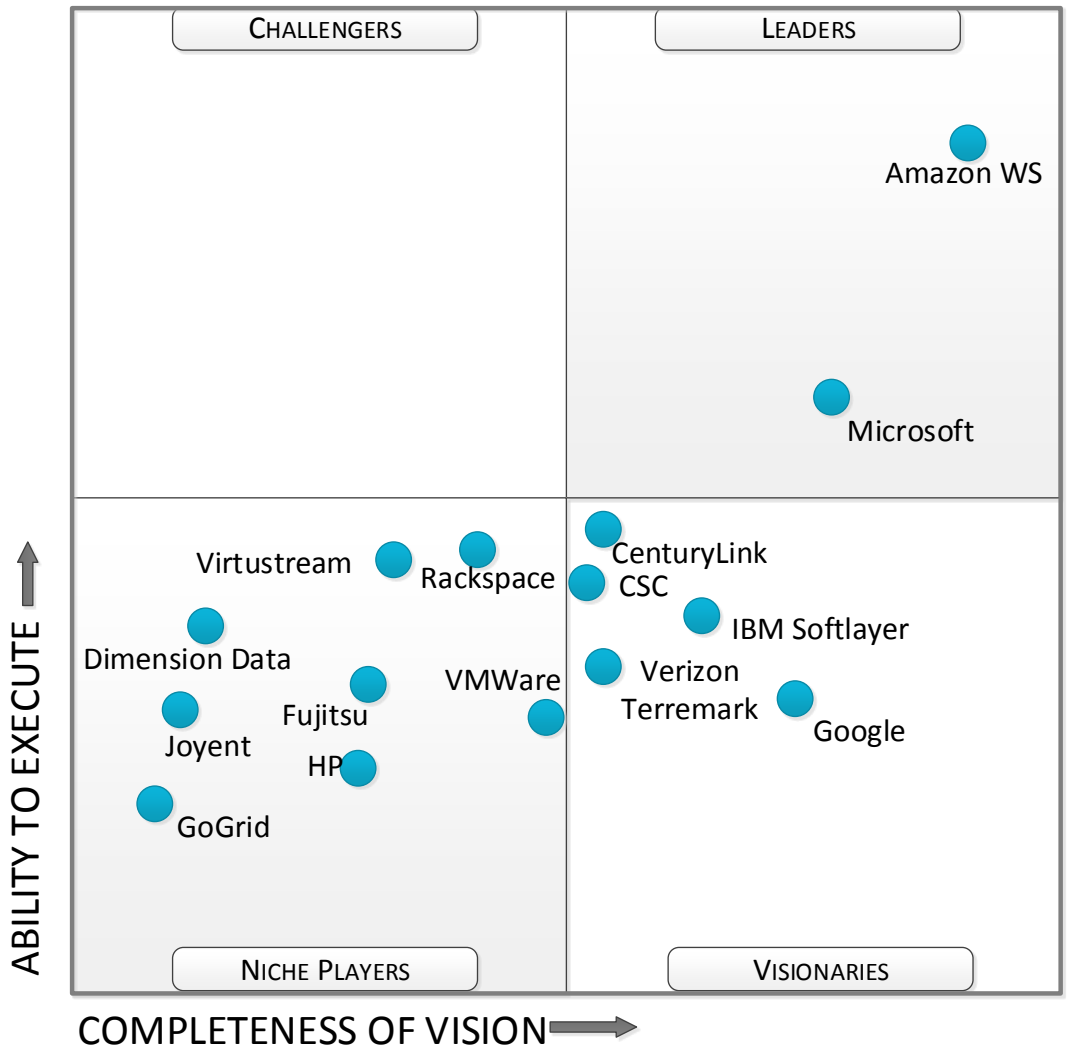

Figure 4.7 - Magic Quadrant for IaaS

Amazon

Although its infrastructure had been used for complex computing, AWS has officially entered the HPC market in 2010 [91] with more tuned hardware profiles, especially targeting HPC. As a well-established Cloud provider, Amazon has been a starting point for many research companies. While AWS offers a good benchmark, its performance is below dedicated clusters [54][55], as expected. However, for medium HPC applications it provides a balanced solution with an affordable price.

Microsoft

Microsoft has made its debut on the HPC market in 2006 with the launch of Windows Compute Cluster Server 2003 [94], exploring a niche of the low-end of HPC market. The peculiar Microsoft model uses an embedded HPC suite, on Windows Server to create a sustainable cluster network. The HPC suite has important components as an application launcher, MS-MPI framework, a job scheduler, cluster management and reporter, all needed to run a cluster efficiently [63]. An also interesting solution is the integration of Microsoft Azure platform, which adds a hybrid layer to burst on demand when more resources are needed. The main solution can also be a problem, as all is provided by Microsoft and other resources, either software or hardware and even different OS like

Linux could not be attached until very recently in April 2013 [95]. Some Unix distributions, like Ubuntu, CentOS and SUSE are now supported.

Google

Google's App Engine (GAE) is the Cloud infrastructure from Google. Unlike Amazon and somewhat like MS Azure, GAE offers IaaS and SaaS but in a SOA manner, through APIs or software deployment. Management, scalability, provisioning and balancing are automatically controlled by Google. The intensive computing product, called Compute Engine, enables fast prototyping and integration of parallel algorithms that are transparently scheduled and executed on the Google infrastructure with a lower resource-provisioning overhead and cheaper than Amazon for jobs shorter than one hour. However, middleware overhead and resource quotas remain the main obstacles that limit its performance [96]. Another problem is that other OS like Windows are not supported.

IBM

CloudBurst is the IBM Cloud platform, which aims to deliver an automatic provisioning of servers, network and storage IaaS allowing the data center to accelerate the creation of service platforms for a wide spectrum of workload types with degree of integration, flexibility and resource optimization [97]. CloudBurst incorporates the WebSphere CloudBurst Appliance, a management suite that supports both physical and virtual nodes. In addition, IBM claims to provide support for public, private and hybrid Clouds as well. IBM has a strong expertise in the HPC field and has some entries in Top500.org, the list of most powerful computers.

Penguin Computing

Penguin Computing business model targets HPC private Cloud solutions [98]. It offers an integrated hardware and software HPC Cloud solution, with a Linux based clustering software that makes a cluster appear and act like a single system, called Scyld ClusterWare. Users have an infrastructure able to execute HPC applications in the Cloud, but maintaining effective control and management of large number of nodes. Using a single point of control, up to 1000 servers can be managed as if they were a single server, resulting on a consistent, virtual system. This simplifies management and deployment, significantly improving server performance and data center resource utilization. The compute nodes are extensions of the Master and run applications specified by the Master. Nodes can be provisioned rapidly, making the cluster extraordinarily scalable and resilient.

CloudSigma

CloudSigma is an IaaS provider based in Zurich, Switzerland. It promises a highly-available, flexible, enterprise-class hybrid cloud servers and cloud hosting solutions [99], both in Europe and in the U.S., implemented with open networking and an open software layers. Its configurations, geographical deployments and pricing models are attractive for HPC users [50].

Rackspace Cloud

Rackspace Cloud was launched in 2006 in the US and it was one of the most successful Cloud providers. Its Managed Infrastructure service level provides a core set of services necessary to set customers up in the Cloud, including architecture advice, security assistance, and code development assistance [100]. Although it does not have any HPC hardware profiles, it claims to fit any reseacher's needs with its custom dedicated datacenter, a heterogeneous HPC environment where faculty, students and others can conduct scientific research.

4.5 HPC in the Hybrid Cloud

Public Clouds are rather immature for HPC, being oriented to business applications [9], even though lately some providers are trying to increase HPC services and support offers [101]. Private clouds emerge with a new managing perspective, controlling different hardware and taking advantage of existing hardware and facilities. A dynamic shared HPC infrastructure offers resource available on demand on pre-defined stack and is much more cost-effective for a university to operate than public

providers [102]. Private Cloud has almost all the benefits of a public Cloud but instead of pay-per-use, costs are shared with other user groups. The environment management is also easier to supervise, because local resources and utilization are known at every instant. Although it can be hard to accomplish, because of network, software and hardware constraints, inside a well-known campus it can be an excellent opportunity to expand and utilize resources to their full capacity.

Adding a public Cloud resource can bring unexpected reaction from IT managers. The perspective of an elastic, pay-as-you-go compute model is extremely attractive on one hand, but on the other, the distress of migration, data privacy and uncertain billing may terrify any responsible. Hybrid deployment comes with another drawback: HPC applications have to be experimented and configured again after the new environment setting.

One of the world's most respected advisory companies in IT, Gartner, has pointed out 5 trends on Cloud Computing strategy through 2015 – one of them refers that Hybrid Cloud Computing must be an imperative [103] – so it shows that the technology has really started to establish as a reference.

Public and private clouds are quickly evolving to enable combined scenarios based on dynamic optimizations for performance and price. Many applications run on public Clouds as it provides an attractive choice to a data center infrastructure. However, for several reasons, applications and data can be forced to be located behind the local firewall, so Cloud investments must be planned to allow a hybrid solution as well. When migrating to the Cloud, the choice is not limited to public or private, since the best of both can be blended. A hybrid Cloud strategy is able to retain high-valued features like cost optimization, service diversity, patterns and risk mitigation although performance is generally lower.

Recent studies indicate that hybrid Clouds are not yet a reality [50], as truly hybrid multicloud scenarios are rare. The tools to manage and seamless move across platforms are not mature, and even between providers using the same underlying Cloud Management Platforms (CMP) there are significant differences in Cloud IaaS implementations.

4.5.1 Benefits and limitations

A hybrid Cloud strategy is able to retain high-valued features like cost optimization, service diversity, patterns, risk mitigation and performance. Some inherent characteristics are detailed in the following paragraphs.

Cost optimization

If on-premises resources are available and execution time is on schedule, saving money is the key. Gathering existing resources on the private Cloud can also ensure a good computing capacity. The inner ability to make adjustments between public and private, balancing price and performance over time is of great benefit.

Service diversity

The hybrid Cloud has an implicit private development, which can provide solutions for heterogeneous services in several areas. When properly designed, the Cloud can also serve user profiles and roles, with particular specifications and requirements.

Risk mitigation

A public Cloud always brings a perception of lost control and can be particularly dangerous when dealing with sensitive data. When providing a hybrid architecture, sensitive data can be deployed on the private side and the remaining application's workloads can be handled on the public side, overcoming the legal and regulatory restrictions.

Performance

While latency of network and storage is generally inferior on private Clouds, public Clouds can deliver greater performance for intensive computing applications because of their continuous state-of-the-art hardware and scalable structure. A hybrid solution must be customized to balance performance, reducing data stream transfer to the strictly necessary. Network traffic on public Clouds is also normally paid depending on the amount of data transferred in or out.

Specialized Management

The large customization offered by private IaaS facilitates the use of any type of hardware to be used to construct a Cloud infrastructure, repurposing other project's resources. The problem of this approach is that it requires a full-time Cloud system administrator, not always reasonable on low budget small deployments [24].

4.5.2 Current players

Apart from this current work and other private educational research, there are not many public records for HPC deployments on hybrid Clouds. Evidently, private developments from commercial firms cannot be discarded but not many have matured enough to go public. However some were recently found.

VMWare

On the HPC field, VMware announced in 2012 version 5 of vSphere as a major breakthrough in terms of performance [104]. Introducing performance counters and enhanced vMotion to replace shared storage, VMWare ensures that virtual memory and disk are copied exemplarily through migrations. Taking advantage of its hypervisor type 1, closer to barebone hardware, demonstrations show that HPC workloads on VMware's virtualization platform achieve close to native performance (in some cases even 20% better than native) with applications from SPEC MPI and SPEC OpenMP benchmark [105]. Their computing infrastructure, VMware vCloud Hybrid Service offers a hybrid IaaS to burst from the private to the public Cloud when needed; the management is made locally on vSphere suite. The problem is that it only offers hybrid Cloud options to existing VMware customers, requiring an on-premises paid IaaS [50].

Nimbix

Nimbix offers a blend of do-it-yourself and Cloud, and promises to address the hybrid requirements in medium to large scale processing environments [106]. Their strategy is to register owned computing resources along with Nimbix HPC datacenters, a process they named "hybridizing". The Nimbix platform aims to eliminate the need to provision systems and instead delivers a unique data processing framework; users only need to point to their data sets, such as read sequence files, select processing pipeline parameters, and submit jobs. All this can be controlled seamlessly by end-users in a web portal, but the process is not disclosed. Cloud compatibility is not shown either, so there is not much to analyze or compare.

Univa UniCloud

Univa delivers the UniCloud solution, providing capabilities for enabling traditional infrastructure to private, public and hybrid HPC Cloud computing environments [107]. UniCloud makes the connection between private and public Clouds, thus supporting hybrid HPC Clouds. It supports a build of several public Clouds, like Amazon, Rackspace and GoGrid with private virtual Clouds based on Oracle VMs and VMware hypervisors. Univa HPC Cloud 3.0, the latest release further enhancements in scalability, manageability and availability areas of HPC solutions running in public clouds.

Microsoft Azure Infrastructure Services

After Azure, previously strictly PaaS, Microsoft launched the Azure Infrastructure Services in April 2013, thus entering the Cloud IaaS business market. For Microsoft customers who want hybrid Cloud

solutions the integration with existing Microsoft tools is particularly appealing. Users can make use of Visual Studio, Team Foundation Server, Active Directory, System Center and PowerShell integration, offering an Azure-like user experience for on-premises infrastructure [50].

HP

After investing resources into its public Cloud over recent years, HP focused their attention on the hybrid Cloud, including private Cloud enablement and Cloud-enabled managed services [50]. As HP's public Cloud offering is based on OpenStack, the open-source IaaS for whom HP is a big sponsor, the hybrid Cloud is completed with support for OpenStack for private Cloud.

Rackspace

Following the general approach in the last years, Rackspace also managed to develop support for hybrid Clouds. Their IaaS is part of a hybrid hosting where Cloud IaaS is additional to a dedicated infrastructure [50]. It also uses OpenStack for the private Cloud.

Chapter 5

5 Sky Computing

The definition of Sky Computing arises as a metaphor to illustrate a layer above Cloud Computing, because such dynamically provisioned distributed domains are built over several clouds [108]. It can be described as a management up-layer of an environment of Clouds, offering variable computing capacity and storage resources with dynamic support to real-time demands. Laying a virtual site over distributed resources, combining the ability to trust remote sites with a trusted networking environment [108], originates a highly elastic response to incoming requests with a seemingly infinite pool of accessible resources. It can be roughly defined as a community cluster. A community cluster is a HPC cluster managed by a faculty group and centrally operated by an institution, maintained for the benefit of the many research groups that own the nodes in the cluster as well as the broader campus community [102].

The motivation behind this work is driven from both the need to offer computing capacity to local researchers and the need to maximize the usage of existing resources. The main objective of this section was to investigate new technical solutions to allow the deployment of intensive computing applications on the Cloud, with the particular emphasis on the private on-premise resources – the lower end of the chain which saves money. The work included experiments and analysis to identify obstacles and technology limitations. The feasibility of the objective was tested with new modeling, architecture and several applications migration. The big challenge here is to manage the combined resources of available Clouds, public or private and interoperate between them. Since the final goal is to support HPC applications, the use of such resources to provide intensive computing in an efficient manner is also a requirement.

5.1 Sky Computing Architecture

To build this architecture we developed a model, able to intersect several functional areas. In Figure 5.1, each Cloud provider has a specific API that makes an interaction available with their own resources. All these APIs can be aggregated by a middleware layer, which allows controlling and managing resources by translating every command to the correspondent provider API. This management is seamless to the end users, as they do not know which resources are being used and where. Abstraction, from bottom to top, is the key for building a consistent system. With an adequate middleware, it becomes possible to use the available resources across any provider and seamlessly increase/decrease resources, according to demand.

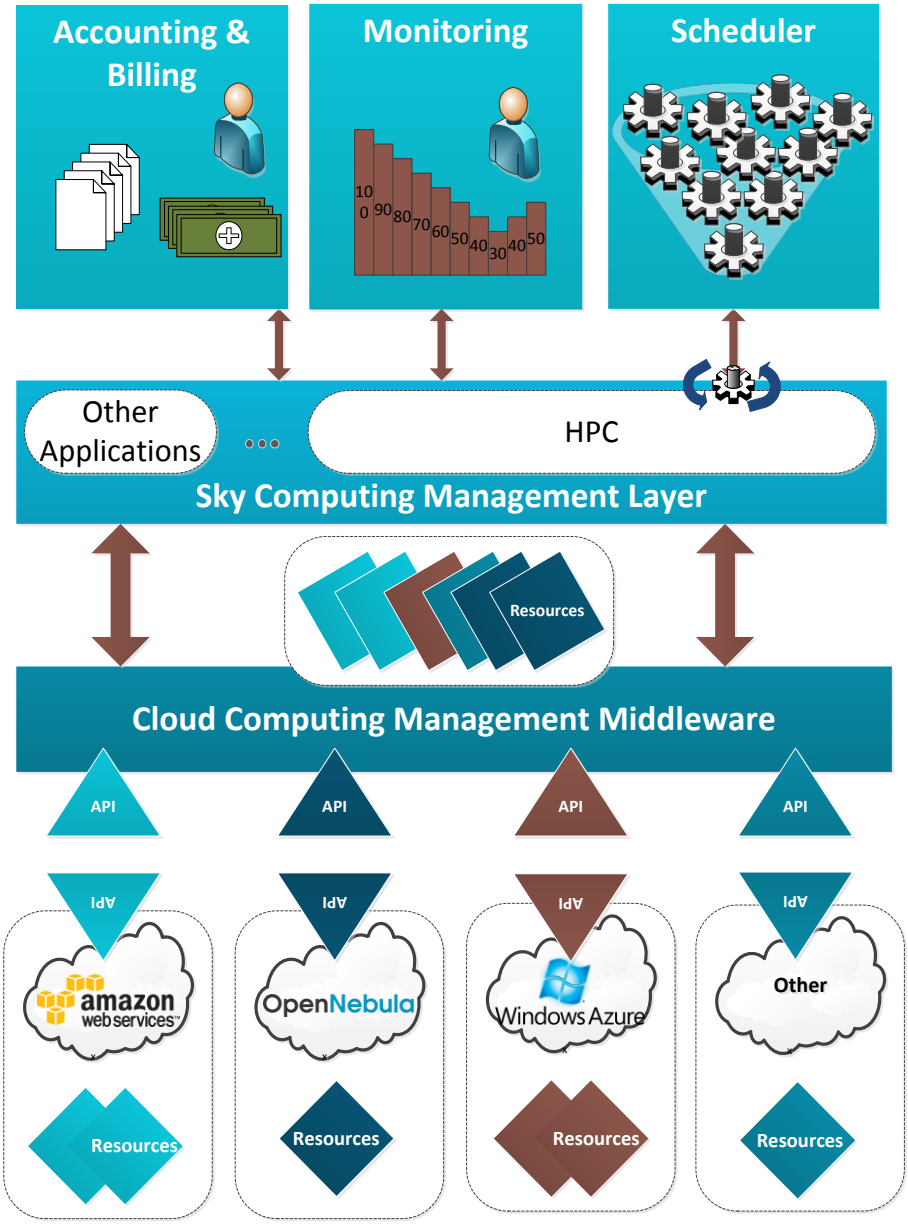

Figure 5.1 - Sky Computing proposed architecture

The upper layer, Sky Computing, integrates the last level of IaaS and the next layer of SaaS. It allows scheduling and distributing resources to inputted tasks/requests made. This is a critical layer, as it must be as comprehensive as possible in features and capabilities; however the result is only a subset or common operations, because the APIs do not support every single operation. Here, our main focus is HPC and intensive computing applications, but it should be as generic as possible to deal with other applications too. Management, with scheduling, accounting and billing, should be thoroughly developed as well as monitoring and job submission.

5.2 Private IaaS

In order to gather and distribute internal resources as a Cloud, we need an IaaS software. There are several approaches to comprehend Cloud Computing but they can be categorized into two Cloud models: datacenter virtualization and infrastructure provision [109], like depicted in Figure 5.2.

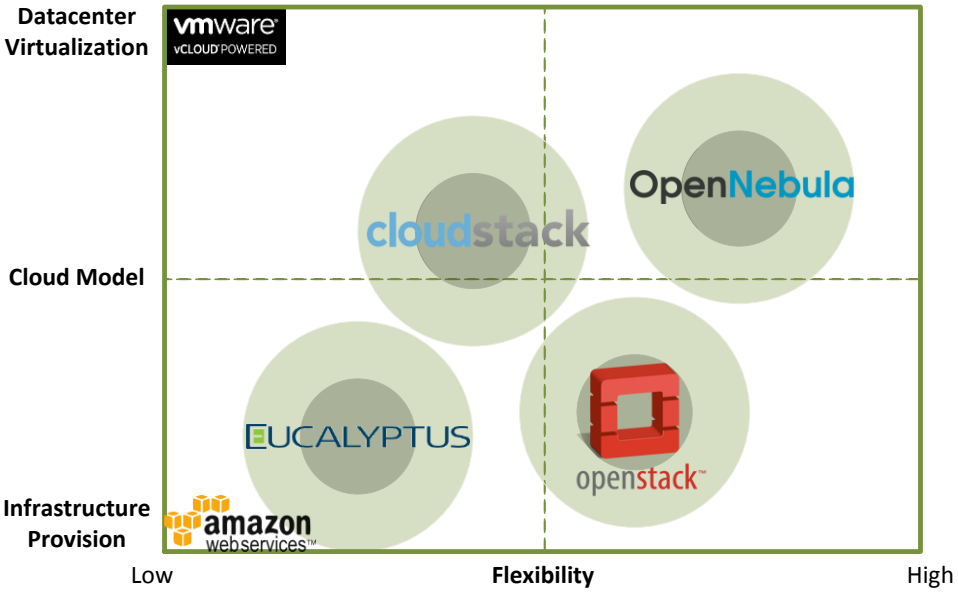

Figure 5.2 - Leading open-source solutions profile [109]

Some companies understand the Cloud as an extension of virtualization in the datacenter and therefore, use infrastructures similar to vCloud to manage and maintain virtualized resources. On the other hand, other companies perceive the Cloud as an AWS-like Cloud on-premise and consequently use provisioning tools to supply virtualized resources on-demand. There are essential differences between all open-source projects, each one has different targets and market purposes and serve dissimilar requirements. Table 5.1 shows the different licensing and business models.

Table 5.1 - IaaS Provider license comparison

Provider	Business model	License
Eucalyptus	Community Edition is free to use Enterprise Edition charges by the total number of processor cores, permanent users need to buy licenses for a specific version	GPLv3 Community version of the GPLv3 license agreement Enterprise Edition uses a custom commercial license agreement, that needs to install the license on the node of the Cloud controller
OpenStack	Free	The community edition uses the Apache 2.0 Apache 2.0 license agreement
OpenNebula	Community Edition is free to use Enterprise Edition repackaging, patches and program access privileges, can more easily be install, configure. Fees are charged by the total number of physical servers and per physical server service at a price of 250 euros per year.	The community edition uses the Apache 2.0 Apache 2.0 license agreement
OpenQRM	Community Edition is free to use Enterprise Edition Community Edition includes repackaging, patches and program access privileges. Users can more easily install, configure and manage a subscription model to provide services. Services start with 1499 euros per year.	The community edition uses the GPLv2 license agreement Enterprise Edition uses a custom commercial license agreement
CloudStack	The Community Edition is free to use. Enterprise Edition provides enhancements and technical support, pricing model is unknown	The community edition uses the Apache 2.0 Apache 2.0 license agreement Enterprise Edition, Citrix CloudPlatform, uses custom license

Because of budget constraints, free use is a requirement for us, which does not mean this architecture cannot be accomplished in another way. Open source is another useful feature, which allows us to access the code and either extend it or adapt it to our needs.

5.3 Middleware Layer

Cloud Computing has become a phenomenon today, amongst every single kind of user and provider. Nevertheless, its disperse development led to an enormous heterogeneity and islets with low interoperability and rare interchangeability, as referred back in 3.6.

Responding to a market demand, most Cloud providers offer their own APIs, allowing users to execute operations from another place. This means Cloud users can use individual providers and still be able to migrate easily to other providers at a later stage, providing a turnkey for migration without the still-in-definition standards. This also opened a slot of opportunity for integrated Cloud management, which can orchestrate resources and deployments of several different clouds.

Middleware is a very important and useful part in the chain value. It provides an abstraction that allows development of applications without being tied to an explicit Cloud vendor. The drawback is that REST API operations are limited (providers' operation set is larger) and can correspond to a loss of performance. The Sky Computing management layer relies on the lower layer resources and interface, so it should be extremely stable and dependable.

There are some projects undergoing for middleware, like the open-source libcloud [110], Deltacloud [111], jclouds [112], or fog [113], while others, like abiquous[114], Kaavo[115] or enstratius[116] offer a more professional customized service and support, in exchange for a monthly fee.

libcloud

The libcloud project is a Python client library for accessing clouds, with a wide range of providers, more than 20. As a library, it relies on other projects to implement a UI, other than command-line. Some of them are not free, as CloudKick [117].

jclouds

As for jclouds, it is also a library targeting portable abstractions for Cloud access. Current version is dating from the end of 2014 is 1.8 and presents a stable CLI implementation – ComputeService and BlobStore, all open-source as well. Unfortunately for us, it doesn't support our private Cloud provider – OpenNebula.
The analysis to libcloud also applies to project fog. It is a Ruby-based Cloud Computing library which allows collections to provide a simplified interface between them. Is doesn't bring a UI attached to view and manage, only a terminal command interface.

Deltacloud

Deltacloud is an open source project developing an API to deal with several Cloud service providers. It targets REST access and backward compatibility across versions, providing long-term stability for scripts, tools and applications. Being a service-based REST API, it works like an intermediary layer, receiving commands and sending them through specially created Cloud drivers, which provide direct operation mapping with the provider's API.

Although there are many implementation differences on these middlewares, they have a similar functioning and way of use. The main difference is on the providers' support and web interface. Table 5.2 shows the differences between these solutions, including also commercial software.

Table 5.2 - Middleware comparison

Middleware	Language	License	Supported providers	APIs	WebUI
libcloud	Python	Open-source	OpenStack, VMWare vCloud StratusLab, IBM SmartCloud	compute, storage, DNS, load balancer	no
deltacloud	Ruby	Open-source	AWS, OpenStack, GoGrid, Rackspace, VMWare, IBM SmartCloud, MS Azure, Eucalyptus, OpenNebula, RHEV-M	compute, storage, load balancer	yes
jclouds	Java	Open-source	AWS, OpenStack, CloudStack, GoGrid, RackSpace, MS Azure, HP Cloud	compute, storage, load balancer	yes (with Maven)
fog	Ruby	Open-source	AWS, OpenStack, CloudStack, GoGrid, IBM Cloud, VMWare vCloud, Oracle VM, XenServer	compute, storage, DNS	no
enstratius	Java	Paid	AWS, OpenStack, CloudStack, GoGrid, RackSpace, MS Azure, HP Cloud, Eucalyptus	compute, storage, load balancer	yes

The requirements for our research were simple: low pricing, release stability and support for existing OpenNebula private Cloud. Based on available middleware, requirements and pricing suitable for the model needs, the Deltacloud project was chosen, including its spin-off project Aeolus [118], a project with open-source tools to build a custom Cloud from both public and private resources.

The Deltacloud API

Deltacloud arises as an elegant solution, integrating libraries, drivers and a Web UI to manage them. It puts strong emphasis on virtual machine templates, or in Deltacloud terms “hardware profiles” as well as the “driver” concept, which are APIs to access Cloud providers.

Architecturally, it follows a slightly different approach from other projects. Unlike OCCI, which focuses on the communication layer, Deltacloud seems to support a 2-layered approach and focuses on the API layer built on top of the communication layer. Figure 5.3 shows the structure of Deltacloud, a SOA (Service-Oriented Architecture) and stateful application.

The Deltacloud server comprehends two key pieces. One of them fulfills tasks like receiving, deserializing HTTP requests and sending responses. The other one contains drivers for specific Cloud providers such as OpenNebula, Rackspace, Amazon EC2, etc. This structure allows to create drivers for new Cloud providers independently and keep working with the whole code.

The REST process is simple, a typical process flow starts with a client request on the WebUI; the core code translates the operation and requests an operation to the selected Cloud provider through its API, remaining stateless. Once the connection is established, the client sends the required authentication data through the HTTP header as part of every request, receiving the server feedback messages, which are translated back into readable information on the WebUI. As a limitation, a single Deltacloud server can only be used to access to a single Cloud and therefore many running daemons are needed to connect to more providers.

An area on which Clouds differ considerably is the available VM hardware. There are many options like memory size, number of virtual CPUs and storage, but a provider might only offer VMs with a fixed 2GB of RAM and 2 CPUs, while others allow the user to set up RAM and CPU within a few limits. Deltacloud uses hardware profiles to manage this area, leveling the providers’ options to a more comparable system.

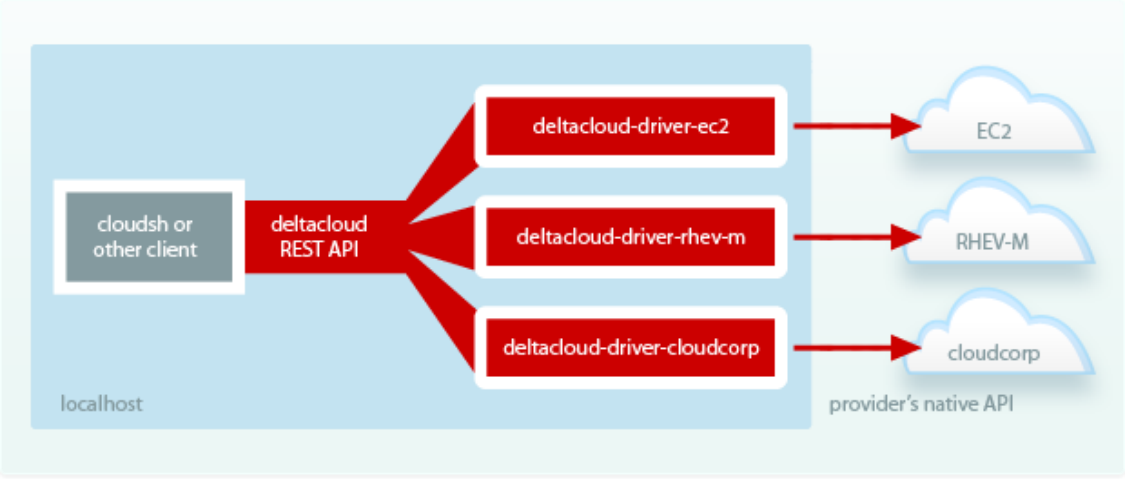

Figure 5.3 - Deltacloud schema [111]

Another area which differs from provider to provider is the VM's lifecycle, which leads to many running states. These states vary not only names, but also in the state sequence and quantity. In order to solve this, Deltacloud implements a finite state machine as a standardized model of the Cloud lifecycle, as shown in Figure 5.4.

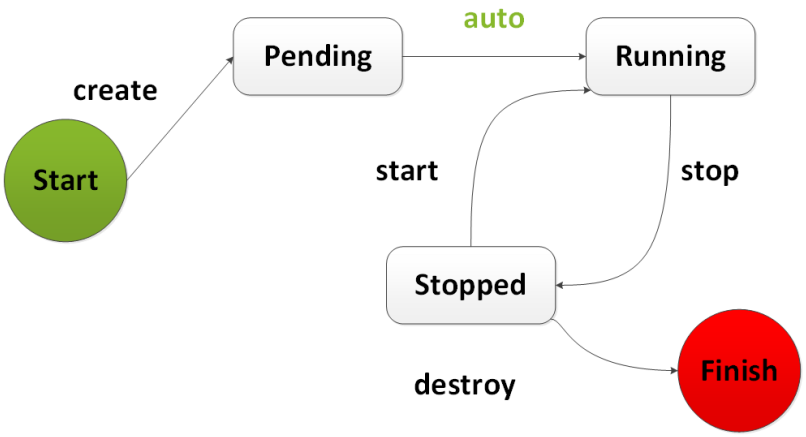

Figure 5.4 - Deltacloud state machine model [119]

Nonetheless, it can be difficult to know exactly which operations (resume, stop) are permitted on a virtual machine, even when the VM lifecycle is known. These operation permissions must be pulled from the providers, which Deltacloud does by indicating for each VM which operations are permitted. Recently, major changes have been made in the project and completion of the main goals is close. The new remodeling set the API UI wrapped aside in a different sub-project named Aeolus. The Aeolus project is now responsible for Web UI, and left the Deltacloud, which focuses on the sole purpose of developing the API core. Aeolus contains a management UI, Conductor, but also aims to simplify tasks - like virtual machine templates - with projects like ImageFactory and ImageWarehouse. In an optimal perspective, a full implementation should permit to seamlessly create/start/run/reboot instances either on private or public Clouds and manage all on the upper layer. So far, supported providers are Amazon EC2, Eucalyptus, Fujitsu FGCP, IBM SmartCloud, GoGrid, OpenNebula, Rackspace, RHEV-M, RimuHosting, Terremark, vSphere, OpenStack, Arubacloud and DigitalOcean.

5.4 Management Layer

The Sky Computing management layer relies on the lower layer resources and interface. Once the middleware API was stabilized and running smoothly, this was the next step to explore. Cloud monitoring and management should be transparent, as well as application deployment. Looking deeper in the proposed architecture, illustrated in Figure 5.5, with some defined agents one can have a better glance of its structure.

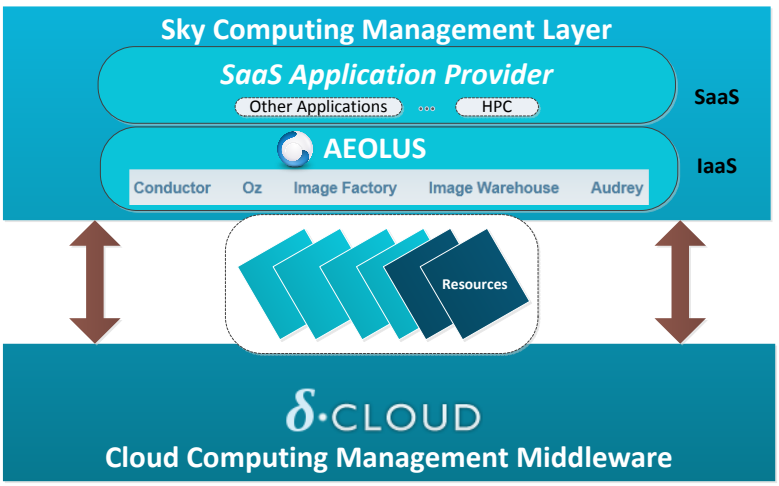

Figure 5.5 - Proposed architecture layer details

Conductor provides a UI interface for managing the available clouds; yet a wider interface is needed in order to allow application deployment and control features like billing and accounting. Clearly this is the SaaS level that needs to be attached. However, higher abstractions than IaaS are off of the Aeolus project. At the moment, current research has not provided a solution that fits this problem and requirements; if not available when needed it will be developed.

5.5 Accounting and Billing

The inherent structure relative to the Cloud premises, either public or private, costs money. On the public side, we have a final cost that reflects the used server's hours multiplied by the hardware profile cost, as seen on Table 5.3.

Table 5.3- Amazon EC2 Hardware Profiles on US East datacenter (August 2014)

Name	CPU	EC2 units	Memory	Storage (GB)	Linux Usage	Windows Usage
Compute Optimized - Current Generation						
m3.medium	1	3	3.75	1 x 4 SSD	$0.070	$0.133
m3.large	2	6.5	7.5	1 x 32 SSD	$0.140	$0.266
m3.xlarge	4	13	15	2 x 40 SSD	$0.280	$0.532
m3.2xlarge	8	26	30	2 x 80 SSD	$0.560	$1.064
Compute Optimized - Current Generation						
c3.large	2	7	3.75	2 x 16 SSD	$0.105	$0.188
c3.xlarge	4	14	7.5	2 x 40 SSD	$0.210	$0.376
c3.2xlarge	8	28	15	2 x 80 SSD	$0.420	$0.752
c3.4xlarge	16	55	30	2 x 160 SSD	$0.840	$1.504
c3.8xlarge	32	108	60	2 x 320 SSD	$1.680	$3.008
GPU Instances - Current Generation						
g2.2xlarge	8	26	15	60 SSD	$0.650	$0.767
Memory Optimized - Current Generation						
r3.large	2	6.5	15	1 x 32 SSD	$0.175	$0.300
r3.xlarge	4	13	30.5	1 x 80 SSD	$0.350	$0.600
r3.2xlarge	8	26	61	1 x 160 SSD	$0.700	$1.080
r3.4xlarge	16	52	122	1 x 320 SSD	$1.400	$1.944
r3.8xlarge	32	104	244	2 x 320 SSD	$2.800	$3.500
Storage Optimized - Current Generation						
i2.xlarge	4	14	30.5	1 x 800 SSD	$0.853	$0.973
i2.2xlarge	8	27	61	2 x 800 SSD	$1.705	$1.946
i2.4xlarge	16	53	122	4 x 800 SSD	$3.410	$3.891
i2.8xlarge	32	104	244	8 x 800 SSD	$6.820	$7.782
hs1.8xlarge	16	35	117	24 x 2048	$4.600	$4.931
Micro and Small Instances						
t1.micro	1	Variable	0.615	EBS Only	$0.020	$0.020
m1.small	1	1	1.7	1 x 160	$0.044	$0.075

On the other hand, on the private side, our architecture allows the aggregation of in-house resources, but they have also costs: hardware amortization, electricity bills (power and cooling) and maintaining personnel.

When providing users with a complex infrastructure like Sky Computing, it is crucial that the right usage is being kept for accounting and billing. Assuring a righteous accounting can predict monthly usage, history analysis and the right planning for future use. The saved data also allows the billing of registered users/research groups/etc. for the used resources, both private and public combined.

Unfortunately, at the time of implementation neither Deltacloud nor Aeolus have accounting and billing implemented, only plans to integrate them on the future. So this part had to be developed, extended from existing feature on IaaS or fetched from another project.

5.6 Monitoring

Monitoring is also a very important part of Cloud management. Probing the resources allows the registration and resource usage control for a healthy running system. For instance, detecting problems (out of memory, power off, CPU overheat, etc.) early for an early resolution.

Neither Deltacloud nor Aeolus have a monitoring feature, so we picked two of the most widely used open source applications for testing, Nagios [120] and Ganglia [121].

Nagios is a monitoring system that enables organizations to identify and resolve IT infrastructure problems before they affect critical business processes [120]. It delivers awareness of IT infrastructure's status and allows the rapid detection of problems, accelerating repair times and mitigating future issues before they affect users. Although Nagios Enterprise charges for support, the solution is open source.

Ganglia is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids [121]. It is an open source solution that uses enhanced algorithms to get very low per-node overheads and is compatible with several OS and architectures.

Both monitoring software were built to monitor physical machines, so we planned to make some developing to adjust them to virtual machines.

5.7 Scheduler

A scheduler is a running daemon that coordinates the virtual requests and the available resources using different scheduling policies. It basically assigns a physical host and a storage area to each VM depending on resource availability, obeying to pre-defined policies.

Once again, neither Deltacloud nor Aeolus have a scheduler, they make the deployment and rely solely on the destination Cloud's scheduler management, which made us search for alternatives.

Haizea

Haizea is an open source resource lease manager and can act as a VM scheduler for OpenNebula or it can be used on its own as a simulator to evaluate different scheduling strategies' performance over time [122]. Haizea supports hardware resources, software environments and manages the period during which the hardware and software resources must be available.

Figure 5.6 is a good example of the workflow of a request on Haizea, customizable with node sizing, processing capacity, available memory and computing time. Upon a request, the scheduler analyzes the hosts' current capacity and deploys the VM if possible.

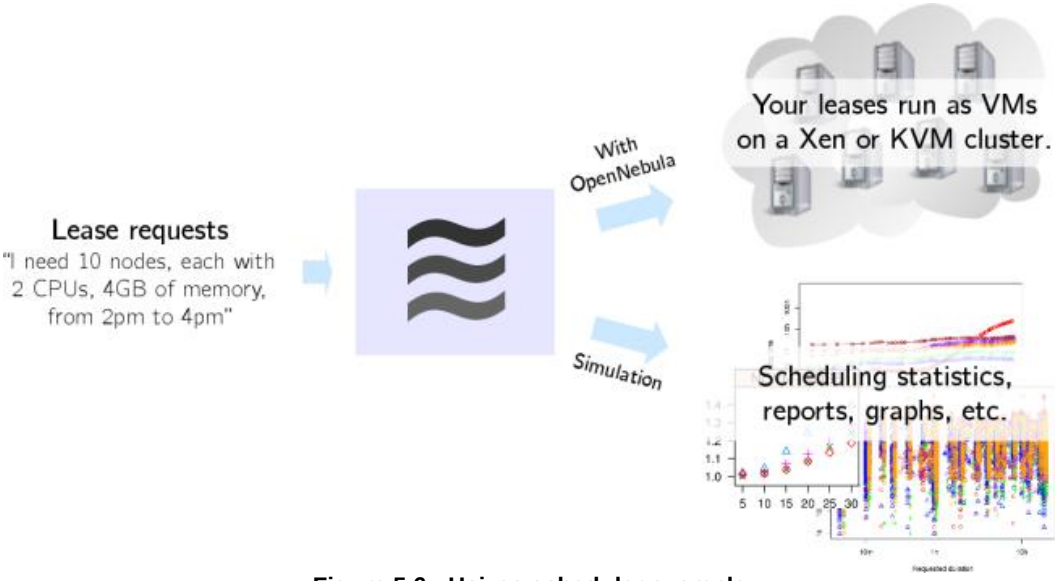

Figure 5.6 - Haizea scheduler example

Cloud Scheduler

The University of Victoria High Energy Physics Research Computing group has an open source Cloud scheduler [123]. This software allows a Cloud-enabled distributed resource manager backend to be attached to an existing IaaS, attached to the Condor IaaS, as seen on Figure 5.7.

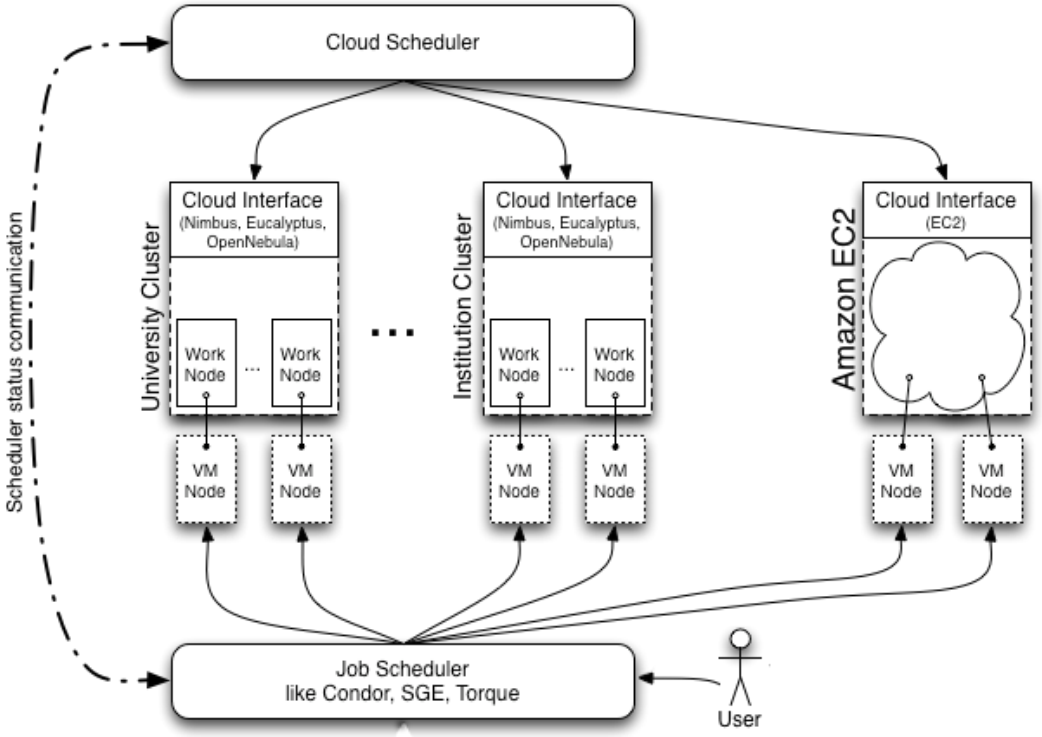

Figure 5.7 - Cloud Scheduler working flow [123]

The process is entirely transparent to the user. After the job submission the scheduler takes control and makes all arrangements to the job's completion. This is even more important when dealing with multiple clouds to get information of available resources and evenly distribute jobs. For instance, the scheduler can move a running VM to another Cloud because other instances of the same job are on that destination Cloud, thus allowing a performance improvement and cost reduction.

In summary, we can say this model can be built with several independent components, which provides the model an interesting module independence. Thus, any replacement can be possible, leaving room for continuous improvement. On the next section we will explore the implementation.

5.8 Setting up a Sky Computing Solution

For executing our proposed architecture, several software solutions were analyzed and studied for a better comprehension and an enhanced support of all features. The steps for building the system are explained on this section.

5.8.1 Development and Implementation

Open-source applications were the main pool of resources, because of their free use. In summary, to complete the whole system we needed:

- a IaaS software to provide management over the physical resources and deliver virtual resources;
- a Cloud Computing middleware that would be able to aggregate Cloud resources from several providers and distribute them over the Cloud;
- a reliable and customizable scheduler, to allow cheaper Cloud bursting on less expensive hours;
- a monitoring software that is problem-aware and able to identify problems;
- an accounting and billing application.

IaaS software - OpenNebula

To support the lower level of the infrastructure, we needed an IaaS software to manage the hardware and support the aforementioned applications. Even limited to open-source applications, there were several available and with a large development over the last few years. Among the most popular, as analyzed on chapter 3, are Eucalyptus, OpenStack, OpenNebula and CloudStack. Each one of them has a different core and programming environment, which allied to specific characteristics make them unalike.

All of them have been evolving and enhancing their extensibility, interoperability and management tools, but they coexist and there is no need to argue about the best, because the best will always be the most suitable depending on the requirements. To choose, we took into consideration and evaluated which components of the framework we needed, as well as the specific public Cloud providers supported and required. Very important as well is the demonstrated compatibility with hypervisors, linking different clouds with unalike hypervisors is a hard piece of the puzzle too.

Due to past experience, tool integration and flexibility we chose OpenNebula. It is an excellent IaaS software, it is open source with a robust and scalable architecture; because of its integration capabilities it is being used by CERN [124]. It has been running on IEETA for a couple of years with proved stability, is compatible with the existing infrastructure, usable with several hypervisors, including Hyper-V, and able to use Amazon EC2.

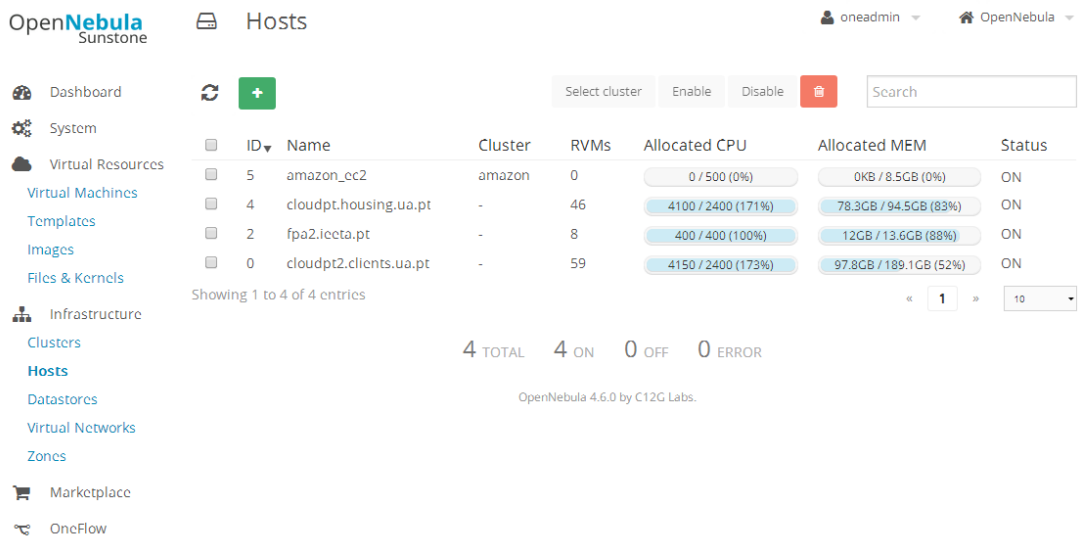

Figure 5.8 - OpenNebula screenshot with available hosts

Middleware - DeltaCloud

Deltacloud has been installed on IEETA's private network. This way we could extend the tests' scope, with both private and public networks and provide a local and stable middleware deployment. After a first version of Deltacloud in 2011, rather incomplete and with some loose ends, the update on 2012 was more stable. Current version is 1.1.3 and has been launched in late April 2013. Deployment wasn't easy, due to several libraries dependencies and incompatible versioning among them. This is one of the major drawbacks of open source: retro compatibility not assured and software which only works with specific library versions. Installation was made on the existing OpenNebula frontserver, inside the university's campus network to avoid firewall access problems.

The research method consisted in successfully install two Cloud API drivers – EC2, Opennebula – and making some tests like deploying virtual machines. The problem here is that for each Cloud, even for two clouds of the same provider, there must be a process running. The solution was to install the Aeolus Project tools, which include Conductor, a multicloud manager.

The install procedure was done by following the instructions of the project. As the OpenNebula server was shared, the installation went on ruby packages instead of the RPM distribution, which is simpler. Deltacloud is composed of two parts: deltacloudd (server) and deltacloudc (client); both were used in the same machine, but could be accessed independently.

The EC2 driver is the default driver for Deltacloud core. By executing "deltacloudd –i ec2" on the server, the Web UI is launched and one can access it on a browser "http://localhost:3001/api" as shown in Figure 5.9.

After login with EC2 credentials, the use of the UI is very straightforward. To create an instance in EC2 one needs to create a bucket first and then create an instance with the one of the available images and hardware profiles. Table 5.3 is an example and refers to Amazon EC2 available hardware to attach to images.

The driver works well and fast enough. However, an intensive workload leads to some wrong information data in instance listing. This is already reported and within development stage; improvements are expected in the next release. The OpenNebula team has contributed with a Deltacloud driver to manage OpenNebula instances through the OCCI API [125]. As OCCI is in version 1.1 and OpenNebula API is also compatible, all restful operations (i.e. create, start, stop, reboot, destroy, list) are currently supported independently of current OpenNebula's new version 4.0. The driver is somewhat more limited than EC2, but regular operations can be carried out smoothly.

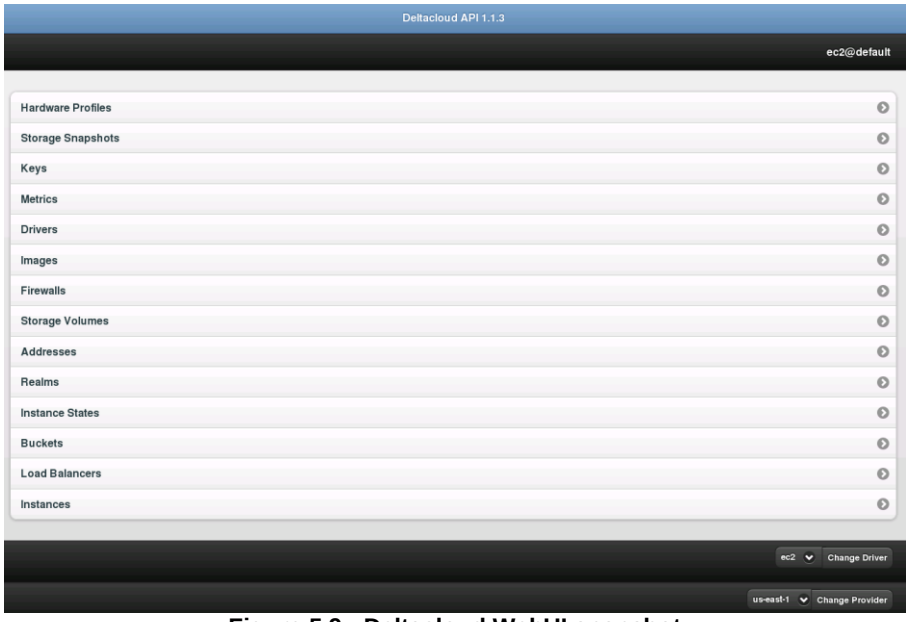

Figure 5.9 - Deltacloud WebUI snapshot

Microsoft Cloud provider Azure has also been tested with a driver connecting to its REST API. Here, as the tests were restricted to storage - the driver still does not support computing – the perspective was to check how the process would flow. The results of the shortlist of operations over containers (creating/deleting) and blobs and (reading/write) were successful.

The Aelous project was the hardest to get to work. Despite its late emerging, it was defined to embrace several projects, rather than just the Multicloud Web UI. It holds 5 sub-projects, where Conductor is the User Interface that links them all.

After an enormous set of tests, and the correct OS settled (Fedora required), Conductor could be deployed successfully. Being a set of WebUIs already tested with Deltacloud, only with some frontpages addition, it lacks a set of desired features. Is has User and Provider Management, a set of the same operations of Deltacloud and it uses powerful libraries as Condor for scheduling and MongoDB as database. Figure 5.10 shows Conductor in detail, in the providers' page. It is almost fully functional, except for the Dashboard page.

Resource management revealed to be very useful; a novelty is ImageFactory, which permits to create templates of images, which can be instantiated on the several available providers at once. At an early development, it can be improved a lot, but it is a great management dashboard for a start.

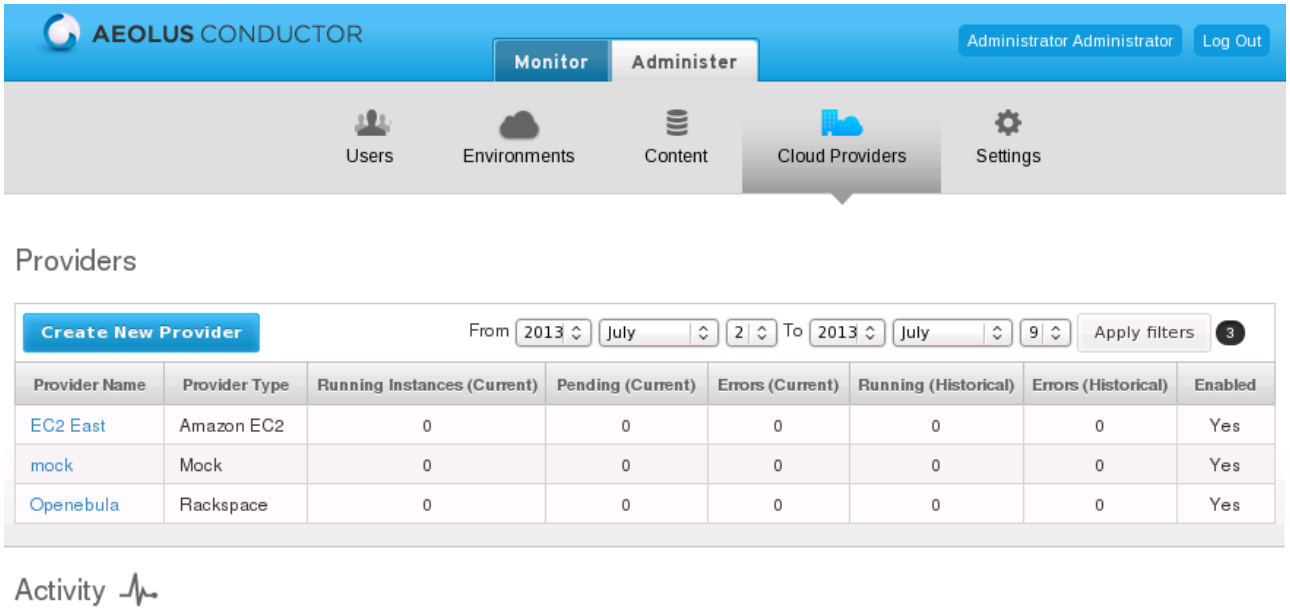

Figure 5.10 - Aeolus Conductor snapshot

Middleware is important and useful. It provides an abstraction that allows developing applications without being tied to an explicit Cloud vendor. The drawback is that API REST operations are limited (providers' operation set is larger) and can correspond to loss of performance with the additional layer.

Accounting and Billing

As there was no open source system available to fulfill this task, we only saved the users' usage on the IaaS database for later handling. This included the number of instantiated VMs (with hardware capabilities), running time and destination Cloud. A simple listing for user 14 on March 2014 shows the output of Figure 5.11.

```
[oneadmin@cloudpt2 ~]$ oneacct  -s "03/01/2014" -e "03/31/2014" -u 14
Showing active history records from Sat Mar 01 00:00:00 +0000 2014 to Mon Mar 31
00:00:00 +0100 2014

# User 14

VID HOSTNAME ACTION  REAS   START_TIME    END_TIME_____ MEMORY CPU NET_RX NET_TX
555 cloudpt _none    none 01/27 16:35:50              -     2G 0.5   6.3G  63.8M
573 cloudpt2 none    none 01/27 16:35:34              -     2G 0.5  10.3G   2.1G
652 cloudpt2 create _user 01/27 15:17:51 03/20 19:16:19  1024M 0.5   3.3G  13.3M
859 cloudpt2 none    none 02/17 18:16:24              -  1024M 0.5   5.4G   6.5M
860 cloudpt2 none    none 02/17 18:13:50              -  1024M 0.5   5.4G   6.1M
861 cloudpt2 none    none 02/17 18:16:24              -  1024M 0.5   5.4G   6.5M
862 cloudpt2 none    none 02/17 18:14:20              -  1024M 0.5   5.4G   6.5M
893 cloudpt_ none    none 02/27 14:34:55              -     2G 0.5   8.6G   2.5G
894 fpa2__   none    none 02/27 14:02:21              -  1024M 0.5 924.9M   2.1M
961 cloudpt2 none    none 03/20 22:04:44              -  1024M 0.5   4.3G   112M
```

Figure 5.11 – User accounting on OpenNebula

Some VM were created before the consulting period, for instance VM with ID 573 was created on January the 27th and shutdown on March the 20th. Notice that there are also registered values for network upload and download, which is very important, especially for public Clouds whose transfer rates are paid.

Monitoring - Nagios

While researching for monitoring, both Ganglia and Nagios appeared to be a good solution. Ganglia was already supported in OpenNebula but it showed some limitations in alerts and reports, so we moved on to Nagios.

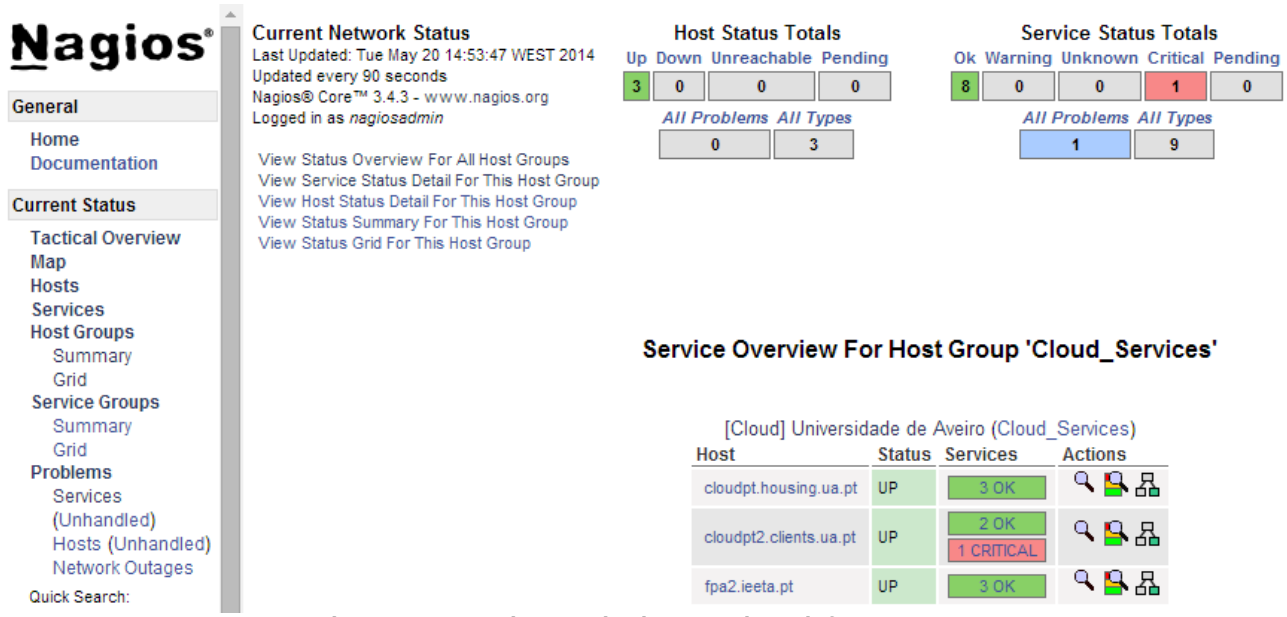

Figure 5.12 - Nagios monitoring on private infrastructure

Scheduler - Haizea

As a scheduler, we chose the Haizea scheduler because of integration as it could directly replace the OpenNebula scheduler. Being open source also widens the opportunity of further development and extension. The installation was straightforward, with selection of a folder and the OpenNebula configuration file. A simple application for a best-effort lease requested at time 13:00:00, requiring 1 hour and 1 node, results on the output of Figure 5.13.

```
[2014-01-25 13:00:00.00] RM      Starting resource manager
[2014-01-25 13:00:00.00] TFILE   Loading tracefile ../traces/sample.lwf
[2014-01-25 13:00:00.00] TFILE   Loaded workload with 1 requests (1 Best-effort)
[2014-01-25 13:00:00.00] CLOCK   Starting simulated clock
[2014-01-25 13:00:00.00] LSCHED  Lease #1 has been requested.
[2014-01-25 13:00:00.00] LSCHED  Lease #1 has been marked as pending.
[2014-01-25 13:00:00.00] LSCHED  Queued best-effort lease request #1, 1 nodes
for 01:00:00.00.
[2014-01-25 13:00:00.00] LSCHED  Next request in the queue is lease 1.
Attempting to schedule...
[2014-01-25 13:00:00.00] VMSCHED Lease #1 has been scheduled on nodes [1]
                         from 2014-01-25 13:00:00.00   to 2014-01-25 14:00:00.00
[2014-01-25 13:00:00.00] VMSCHED Started VMs for lease 1 on nodes [1]
[2014-01-25 14:00:00.00] VMSCHED Stopped VMs for lease 1 on nodes [1]
[2014-01-25 14:00:00.00] VMSCHED Lease 1's VMs have shutdown.
[2014-01-25 14:00:00.00] CLOCK   Simulated clock has stopped
[2014-01-25 14:00:00.00] RM      Stopping resource manager gracefully...
[2014-01-25 14:00:00.00] RM      --- Haizea status summary ---
[2014-01-25 14:00:00.00] RM      Number of leases (not including completed): 0
[2014-01-25 14:00:00.00] RM      Completed leases: 1
[2014-01-25 14:00:00.00] RM      Completed best-effort leases: 1
[2014-01-25 14:00:00.00] RM      Queue size: 0
[2014-01-25 14:00:00.00] RM      Accepted AR leases: 0
[2014-01-25 14:00:00.00] RM      Rejected AR leases: 0
[2014-01-25 14:00:00.00] RM      Accepted IM leases: 0
[2014-01-25 14:00:00.00] RM      Rejected IM leases: 0
[2014-01-25 14:00:00.00] RM      ---- End summary ----
```

Figure 5.13 - Haizea scheduler log

Since the cluster is empty, the request is scheduled immediately. The VM lease starts at 13:00 and stops at 14:00, following the 1 hour application.

5.8.2 Assembling the System

The hardest part was to connect all pieces of the puzzle. We managed to get Aeolus working with a hybrid infrastructure, featuring Amazon and OpenNebula with a custom scheduler Haizea [122] and Ganglia over a CentOS 6.2 OS. Figure 5.14 illustrates the executed implementation.

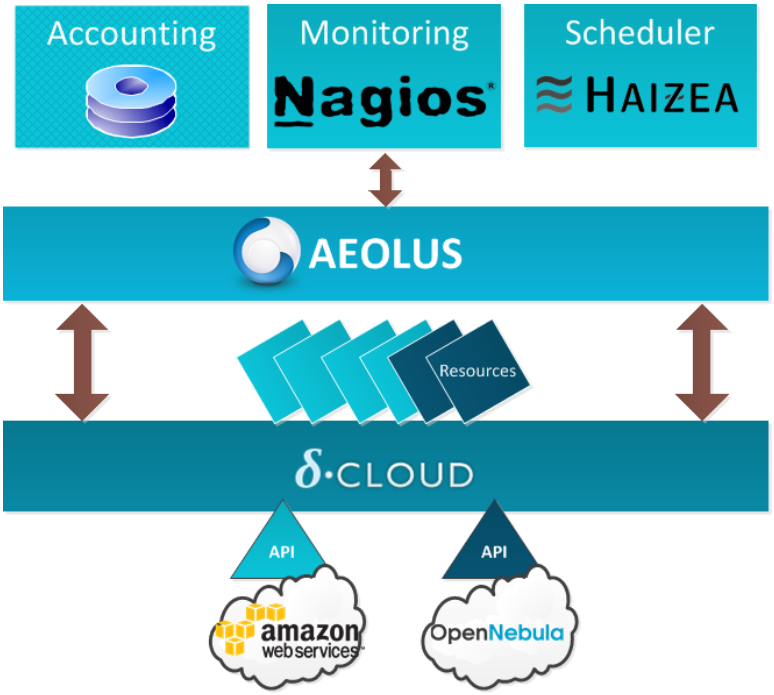

Figure 5.14 - Implemented architecture

The structure was functional and stable, however the installation of an accounting system was not possible.The lack of some important pieces reduced the structure flexibility and agility, despite the occasional improvement by a new tweaks on fresh software updates.

Meanwhile, some open source projects have decreased their developers' task force – Deltacloud (2013), Aeolus (2013) and Haizea (2011) – and despite being held by the community few advances are expected. This was a major drawback for the ongoing work.

On the other hand, private IaaS have grown stronger and added many functional features as scheduling, monitoring, accounting and APIs delivering Cloud interoperability. The logical step was to select a powerful and capable IaaS, enhancing all needed features.

Nevertheless, we managed to build a functional architecture. Instead of stating which hardware to lease out, when and for how long, along with its network flow problems, there is always hardware available and a local network whose bandwidth can be customized. We included three servers into our private IaaS, building a small-sized structure, totalizing 52 CPU cores (with hyper-threading) and 300GB of RAM. As soon as the rest of the system was implemented, the next logical step was to experiment the executed system in order to test the model with a real approach.

Chapter 6

6 Application Scenario

Migrating traditional HPC applications to the Cloud is not a straightforward process. Running systems need to be virtualized to create templates, operating systems need tuning with the new infrastructure and software requires adaptation to Cloud specifications. However, Cloud Computing offers many potential benefits to HPC developers and users. It facilitates dynamic acquisition of computing and storage resources and access to scalable services; moreover, Cloud platforms abstract away the underlying system and automate deployment and control of supported software and service [24]; such configurations can last more than a few weeks in traditional datacenters. In addition, they have a very balanced cost-performance ratio [126], though they are not yet able to compete with traditional HPC supercomputers [127]. The Cloud holds great promise for the scientific computing community as it can be a cheap alternative to supercomputers and specialized clusters, more reliable platform than grids, and more scalable platform than the largest commodity clusters [128].

In UA, the university campus gathers 15 faculties plus another 14 research labs and one IT center. Many of them have powerful hardware to accomplish some projects' tasks, but are scattered among different data centers and may be seen as behind the curtain – they're not aware of each other. Within a Cloud IaaS implementation, resources can be aggregated from all over the university campus, being available to whoever needs them and thus providing a more powerful and seamless unique platform.

In order to accomplish the proposed objectives, finding application scenarios was an essential step both for the infrastructure test and the feasibility analysis. These scenarios will provide:

- The study and tuning for research applications, which may result in innovative solutions;
- The challenge of data management with network contraints between Clouds, which will produce the intended workload mobility;
- The deployment of applications on hardware, giving deep insight to create a custom solution;
- The continuous use and users' feedback that provides an invaluable asset to build a management base.

The implemented system was used for deploying several applications, such as weather forecast, genomics processing and lab classes' support. On the area of scientific examples we used, for instance, the Weather Forecast application from the University of Aveiro's (UA) Department of Physics (using WRF [129]) and the 1000 Genomes Project [130] that was being embraced by the Bioinformatics Research Group. Each application was standardized to fit on a virtual machine template and ran virtualized on our infrastructure.

IEETA has a private Cloud installed on three blades with 28 CPU-cores, running OpenNebula and hosting other testbed services. Our expectation was to use this cluster to aid intensive computing, enhancing the results, and then extend the private Cloud to use other low-operating clusters, spread

all over by the campus, or the public Cloud when needed. With the right management and service level agreement, the whole university would largely benefit from it, both in time and money. The implementation of this hybrid applications' scenario is detailed in this chapter.

6.1 1000 Genomes

One of the first preliminary tests was the 1000 Genomes application. The project aims to sequence the genomes of a large number of people, to provide a comprehensive resource on human genetic variation [130]. The main goal of the 1000 Genomes Project is to find most genetic variants that have frequencies of at least 1% in the populations studied, so that diseases and other disturbances can be pre-detected and dealt with in a timely manner. This goal can be attained by sequencing many individuals lightly.

The challenge given to us was to optimize the computing process. Traditional approach follow a linear processing algorithm. This process was being run on a powerful, but shared physical machine. The algorithm is outlined on Figure 6.1.

```
for each chromosome
  split chromosome .vcf into 1092 individual sample .svcf files,
                        one group at a time. (Group size is a PARAMETER!)
  for each sample
   patch the reference/chromosome.fa to generate the sample/chromosome.fa
        count words in sample/chromosome.fa
  end
end
```

Figure 6.1 - 1000 Genomes pseudo-code

So, each chromosome (1 to 22) of every of the 1092 individuals had to be split, referenced and samples counted, as in `new_script.sh`, on Appendix A3. The problem was adapted to MPI, which does not need to recompile the programs to run in parallel. We launched the `mpiexe.hydra` executable with 22 processes, each one on a CPU core. The results for a single chromosome on 1092 individuals are shown on Table 6.1, with estimation price (in US dollars) of public and private Cloud.

Table 6.1 - Processed data for chromosome 22 on 1092 individuals

Scenario	$/h	$ total	Elapsed Time
Original	$0,00	$0,00	41:04:53
Private Cloud	$0,00	$0,00	03:53:15
Public Cloud	$2,40	$12,00	04:18:54

As preliminary results were very good, the task to continue was let to the Bioinformatics group. The main reason of the good results was the MPI compilation which parallelized the process in the existing cores. Such parallelization can improve computing time, but on the other hand it can also mean a cost increase, especially on the public Cloud where every traffic is charged.

6.2 CLiM@UA

Modern weather forecast models, in contrast with early models that were essentially statistic, are produced by numerical weather prediction systems, with sophisticated data assimilation systems for determining the initial state plus advanced high resolution dynamical global or regional models of the atmosphere [131]. These numerical simulations were one of the first applications of HPC remaining one of the most important today in terms of impact and relevance to the public [132].

The satellite technology advances throughout the last decades resulted in rich data acquisition about the Earth's surface. This data, with great resolution and detail, can be used in numerical atmospheric models, thus allowing the simulation of the lower boundary forecast. The permanent development in the computing industry allows finer resolutions, larger domains and more complex models.

Meteorological models are an extremely powerful and useful tool to analyze and simulate weather patterns. One of the most used tools is the Weather Research and Forecasting (WRF) Advanced Research (ARW) solver, a widely used community mesoscale model. It represents the current state-of-the-art in mesoscale model development, and was established as a successor to the long-standing Penn State/NCAR Fifth-Generation Mesoscale Model (MM5), sharing much of the same dynamics and model physics [133]. The WRF modeling is applicable for a variety of areas, including storm-scale research and prediction, air-quality modeling, wildfire simulation, hurricane and tropical storm prediction and regional climate.

The performance and reliability of these numerical systems are very important assets, both for people and industry. They help not only to predict when and where it will rain but also the tides and the wind speed for electricity production.

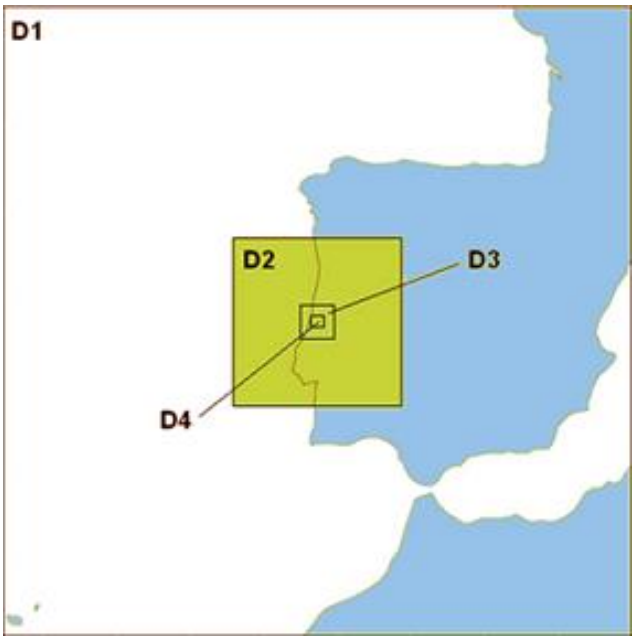

Figure 6.2 - Nest simulation domains [133]

The WRF model is based on geographic slices, called domains. Each domain can have sublevels of domains, named nested domains. Nesting is a form of refinement that allows costly higher resolution computation to be focused over a region of interest [134]. For instance, on Figure 6.2 the WRF model is built over a parent domain (D1) with 90 km of spatial resolution, covering all of the Iberian Peninsula and a portion of the North-Western Atlantic Ocean. The first nested domain (D2), with a spatial resolution of 18 km, comprises the Northern and Central part of the Portuguese territory. The innermost domain (D4) has a spatial resolution of 3.6 km and it is focused on the chosen area to simulate, located in central Portugal [133]. WRF has three layers, Driver, Mediation and model. The Driver layer is responsible for run-time allocation and parallel decomposition of model domain data. The Mediation layer incorporates one time-step of a particular dynamical core on a single model domain, using at a first step MPI, for communication and multithreading, and OpenMP on a second step, for domain decomposition within distributed memory. The Model layer encompasses the actual computational routines that form the model: diffusion, physical parameterizations and advection, among others [134].

One of the existing projects led by Department of Physics consists in the daily weather forecast performed by CliM@UA [135]. The project is a decision support system to the national wind generating power, one of the biggest in the world [133]. Combining the statistical estimates with the weather forecast, one can calculate the power needs for each time span. According to the power generating forecast, it is possible to decide which alternative to use: thermoelectric stations, dam stations or even importing from nuclear power stations on Central Europe.

The weather forecast tasks, performed by CliM@UA started in 2007. At that time, the system was installed on a 2 CPU-core with 3 GB RAM and OpenSuse 10.2. The system performed four daily forecasts over Portuguese mainland, Azores and Madeira archipelagos. Using the version 2.2 of the WRF-ARW model [129], a much simpler version than the current 3.3.1, the system had to rely on sparse horizontal resolution in order to perform the task. Nowadays the system is much more complex and heavier in terms of computational resources.

The weather forecast is performed four times per day, resulting in a 7 day forecast, 6 hours ahead from the original global model. The process starts with the download of the analysis and forecasts data at the synoptic hours - 00, 06, 12 and 18 Coordinated Universal Time (UTC.) Every day, weather data from the Global Forecast System (GFS) model [19] is downloaded to its servers, cropped to specific geographical areas and then heavy computation is done to forecast fields as humidity, temperature, rain, fog, among others. After successful download, the WRF Preprocessing System (WPS) converts the data which is then cropped to the geographic area of interest, as shown in Figure 6.3. At the moment the forecast uses two nested domains for Portugal mainland, a sparse one with 25 Km of horizontal resolution and a finer one with 5 Km. Madeira and Azores also have nested grids with the same approach. In Azores due to the huge area involved one sparse domain is used with 25 Km and then 3 finer domains with 5 Km are used for the east, center and west parts of the archipelago. The WRF model then uses the data prepared by WPS as input and boundary conditions and begins the simulation for Portuguese mainland, Azores and Madeira archipelago

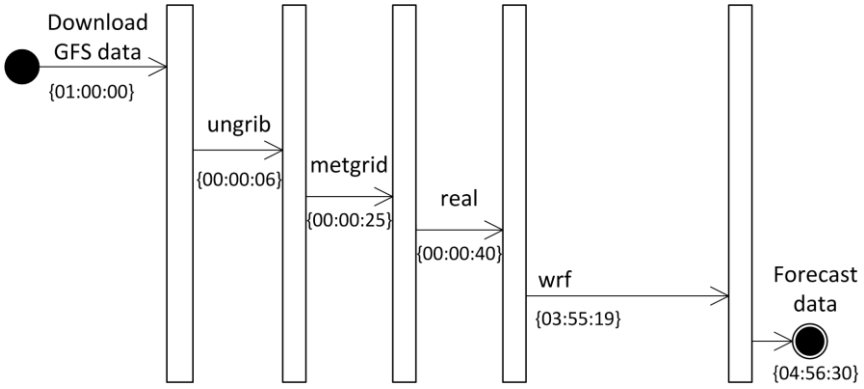

Figure 6.3 - Forecast process workflow for 6h in Portugal Mainland

As shown in Figure 6.3, after GFS data download from US servers, the preprocessing begins. The forecast data is decoded from the `grib` format into an intermediate format - which includes version number, gridded data common information, particular grid type specific information and a 2-dimensional slab of data - with the `ungrib` application. Then, the `metgrid` executable horizontally interpolates the data and vertical interpolation is made by the `real` program.

The regular forecast performed by CliM@UA requires 12 CPU-core for Portuguese mainland, 4 CPU-cores for Azores archipelago and 2 CPU-cores for Madeira archipelago. With the available hardware solution, there is a marginal window of a few minutes per forecast. Considering this window, finer scales are not feasible, since the overall forecast would be ready past the beginning of the expected forecast. Another 6 CPU-cores are used to perform weather forecasts to Romania and another 12 CPU-cores in Angola, but these are still experimental.

HPC is one way to perform this task, recurring to MP with parallel processing. Currently this laboratory is using a head node with 4 CPU-cores and 4 physical slaves, each one built on 12 CPU-core at 2.66GHz with 12GB RAM for its projects and tasks. For the weather forecast, a head node and a slave are normally used, but one more slave may be obtainable in the occurrence of unexpected events like power outage (nature phenomena) or storage faults. This is enough to slow down forecasts on our tight 6h window because it would be unsuitable to forecast the 0am to 6am period at 5am. Plans are being made to implement a new setup of the model, which will be adjusted to the cluster's capacity. Even now, the forecast horizontal scale is 5 Km, because with higher resolution there are not enough computing resources to deliver the results on time. The emerging of

the Cloud on the HPC area was an excellent opportunity to take advantage of its inherent characteristics as elasticity and scalability.

Figure 6.4 displays some data analysis statistics while simulating, showing CPU usage is very high at the interpolation phase.

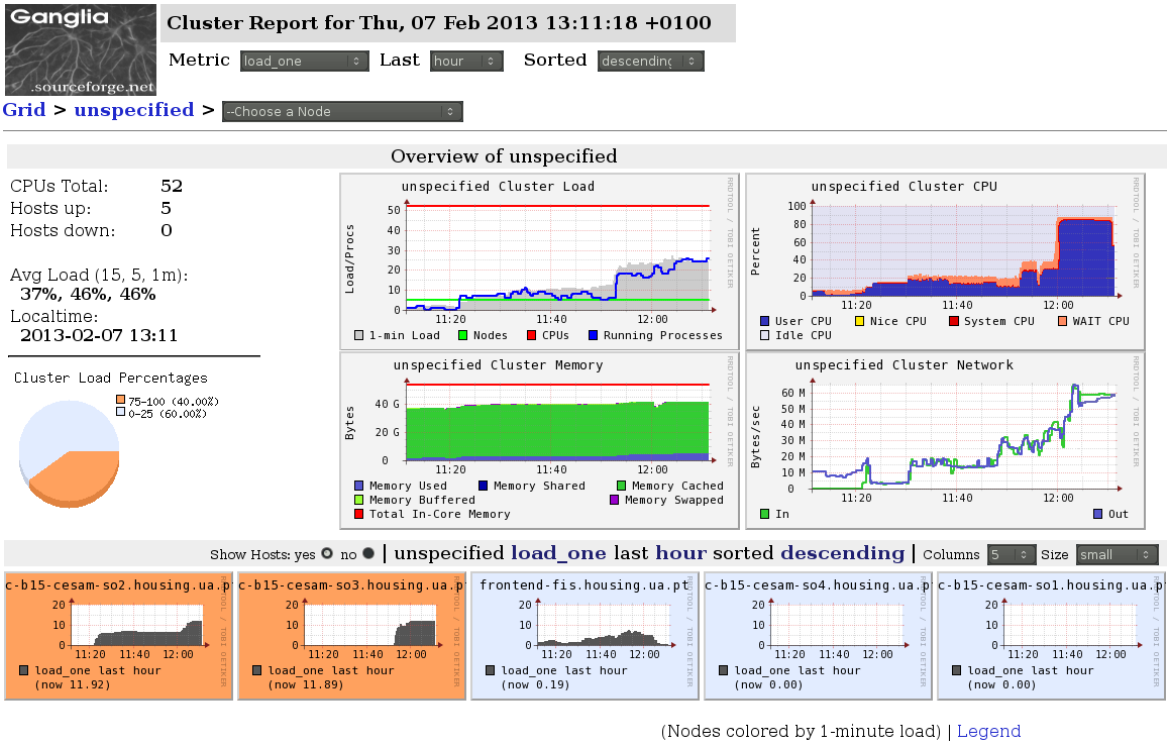

Figure 6.4 - General view of the cluster 1 min. load status during simulations

This was the initial challenge in migrating traditional HPC to the Cloud, a real problem with some issues to be solved. Starting from scratch, the plan was to mount an IaaS able to manage hybrid resources and deliver good results.

Our motivation was to enhance this real application with constraints, which was the initial challenge in migrating traditional HPC to the Cloud. Starting from scratch, the plan was to mount an infrastructure able to manage hybrid resources and deliver good results.

However, it is critical for exposing and improving the performance defects of virtualization systems to measure and compare their processing efficiency [14]. The computational power of the combined slaves is not directly comparable with the traditional cluster because of the overheads inbetween all the hardware – virtual or not. So, these faults must be minimized to make the model work and make it viable.

Another important aspect is proactive performance management, which requires predictions of the application-level performance under varying service workloads [136]. The possibility of burst to public Cloud is also a key advantage to this project, because it can have means for completion when some unexpected event, causing delays, occurs.

The current approach is limited to the hardware capacity. The virtual cluster provided through the Cloud can extend beyond its outer bonds, which is especially important to prevent unforeseen events and avoid delays of the processing results.

6.2.1 Proposed Model

Using a template is an excellent way of standardizing a virtual machine to replicate on clusters. Once it is finished, parameters such as CPU or RAM can be adjusted on the template's configuration to adapt to cluster specifications. This way, one can have 4-core/4GB on some machines, and 8-core/4GB RAM on others, changing only these two hardware specifications on VM creation.

Our structure is composed by two VM templates: a first template containing one head node, with service configuration, and a second template for slave worker nodes, all with the same node configuration. This structure can be easily redefined to include several head nodes with different configurations or different sets of worker nodes. These sets of working nodes can be characterized by computing capacity – it is mandatory to find suitable common hardware configurations. More modest machine configurations are able to work in smaller hardware, thus taking advantage of more machines, and bigger configurations which allow minimizing VM overheads and delays.

As referred before, OpenNebula was used to provide the IaaS, installed on a Scientific Linux 6 OS. The used architecture is depicted below on Figure 6.5:

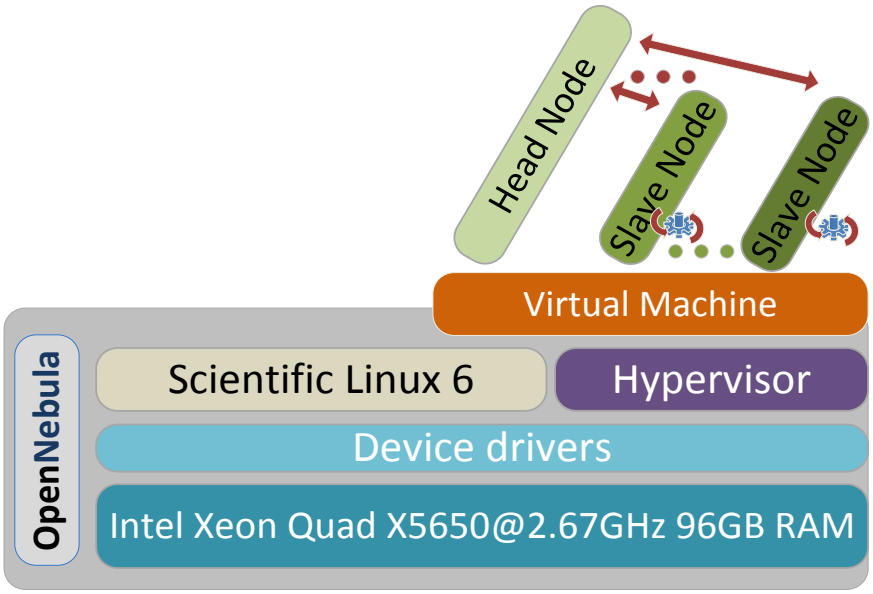

Figure 6.5 - Proposed architecture

OpenNebula supports many Unix distributions, but Scientific Linux was chosen for being a solid and stable distribution which can make a reliable core operating system, with a relatively small system footprint.

The used model is depicted in Figure 6.6, where we can see the private IaaS to instantiate these VM templates, with the optional deployment on the public Amazon Elastic Compute 2 (Amazon EC2). EC2's choice was related to the single existing API for public providers held by OpenNebula. No price comparison and cost evaluation were made at this time, EC2 was the starting point to allow smooth usage, based on its big market share.

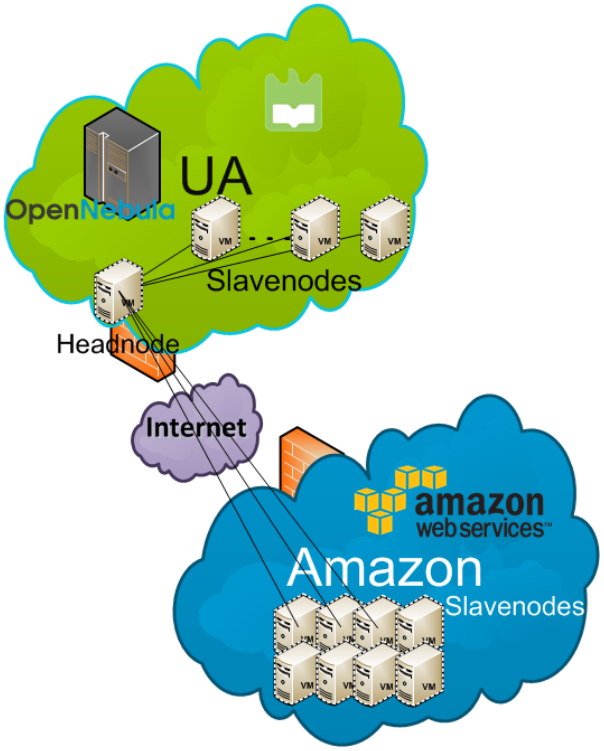

Figure 6.6 - Proposed model

6.2.2 Configuring Virtual Machines

Starting the task to improve results, we made a template of head node in OpenNebula and another for slave node. The head node template included:

- 70GB of hard disk,
- 2 GB of RAM
- 2 CPU cores
- 2 network cards

The slave node template included:

- 20GB of hard disk
- 4GB of RAM
- 4 CPU cores
- 2 network cards
- Startup script to configure VM

This machine setup was driven from the original configuration. The process was to replicate the initial schema, leaving enhancements for a later stage. The allocated extra disk for the head node is needed for the weather forecast raw data download and post processing; to schedule jobs and manage slave nodes, 2GB of memory and 2 cores is enough.

As for the slave node templates, we started with 4GB of memory and 4 cores, which should be able to fit on most of the clusters' hardware specification. The processed results are saved in a NFS folder, shared by the master. The OpenNebula template is detailed on Figure 6.7.

```
CONTEXT=[
CORES=$CPU,
FILES="/opt/opennebula_shared/context-scripts/centos6/id_rsa.pub
/opt/opennebula_shared/context-scripts/centos6/init.sh
/etc/resolv.conf",
  GATEWAY=192.168.1.254,
  GATEWAY_PRIVATE=10.1.1.0,
  GATEWAY_PUBLIC=192.168.1.254,
  HOSTNAME=climetua-$VMID,
  IP_PRIVATE="$NIC[IP, NETWORK=\"HPC Private Network\"]",
  IP_PUBLIC="$NIC[IP, NETWORK=\"HPC Network\"]",
  NETMASK_PRIVATE=255.0.0.0,
  NETMASK_PUBLIC=255.255.255.0,
  USERNAME=theusername,
  USER_PUBKEY=id_rsa.pub ]
CPU=4
DISK=[ BUS=virtio, DRIVER=raw, READONLY=no,
  SOURCE=/mnt/data/one-var/silver-images/CentOS-Node.img, TARGET=vda ]
DISK=[ BUS=virtio, SIZE=1024, TYPE=swap ]
FEATURES=[ ACPI=yes ]
GRAPHICS=[TYPE=vnc ]
MEMORY=4096
NAME=CentOS6-Node
NIC=[ MODEL=virtio, NETWORK_ID=12 ]
NIC=[ MODEL=virtio, NETWORK_ID=7 ]
OS=[ARCH=x86_64, BOOT=hd ]
RAW=[TYPE=kvm ]
TEMPLATE_ID=11
VCPU=4
```

Figure 6.7 - OpenNebula Virtual Machine Template

After a clean install of CentOS6, all required software was installed: Fortran and C++ compilers, MPI, netcdf, zlib, jasper and libpng libraries were compiled. Once all requirements are fulfilled, the Weather Research and Forecast model (WRF-ARW) version 3.3.1 was installed [129]. The model installation requires two major steps, first the compilation of the WRFV3 model and then the WRF Preprocessing System (WPS). The WPS role is to prepare the data, so that it can be used by WRF.

The head node is available by two NIC attributes, one of them using a pre-configured static and public address, created on VLAN HPC Network, the other sharing the subnet of the worker nodes called HPC Private Network. Information about shared disk partitions is also already configured. The head node maintains on `/etc/hosts` and `hydra.pt` files information about the slaves' IP address and CPU cores. Information on the head node is crucial to make the best schedule for the jobs.

OpenNebula allows automatic VM configuration, which is called contextualization. The files included in the VM's template have information and scripts which the machine will run on the first startup. Hostname, network configuration, allowed hosts, SSH keys, among others, make the VM quickly configured and available.

6.2.3 Deploying Virtual Machines

The head node shares the subnet of the worker nodes and information about shared disk partitions is also already configured. The head node maintains on files `/etc/hosts` and `hydra.pt` information about the slaves' IP address and CPU core number. Information on the head node is crucial to make the best schedule for the jobs. The creation of slave nodes is very quick due to the use of qcow images – copy on write from QEMU [137] – that leave the main file intact and create a separate one with only the changes (the write part). Since in this case the operating system image is not going to be changed (all data is handled on a different, shared location), we may benefit from

qcow2 to keep the disk footprint to the lowest possible. A couple of minutes are enough for a new VM instantiation and initial script for network configuration, while a server installation would require a couple of hours.

To deploy a new VM, we can create it with from the template OpenNebula in terminal or by a browser application Sunstone. Figure 6.8 shows the simple process.

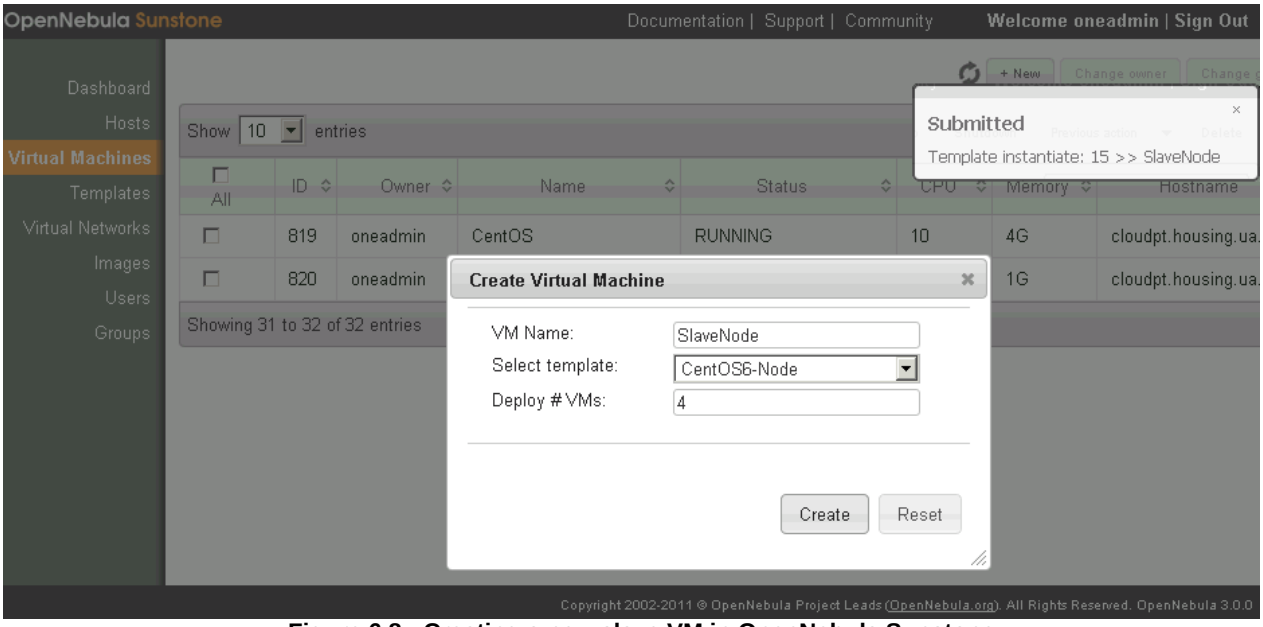

Figure 6.8 - Creating a new slave VM in OpenNebula Sunstone

When a new slave node is created, a boot script is executed to configure hostname, network, NFS sharing and update the necessary files on the head node with the node information.

The transition from the traditional to a new Cloud approach was made smoothly, even with some tuning to be done, for instance in OS, network and storage. The used network was the UA Campus network with no defined Quality of Service (QoS), all data went through the Infiniband channel with all other traffic. As for storage, the master node was running NFS and sharing the folder with the slaves, but a centralized dedicated server (not virtual) with NFS or other technology like Gluster [138], MooseFS [139] or Lustre could be helpful too.

6.2.4 Deployment scenarios

To experiment a wide range of options and variables, 6 scenarios were built changing master, slave and shared disk locations. This gave us the opportunity to test public, private and hybrid Clouds on simulations.

To test the possible scenarios, we used the local machine with OpenNebula as private provider and Amazon Elastic Compute 2 (Amazon EC2) Cloud as the public provider. After building the master and slave templates, both on OpenNebula and Amazon EC2, we instantiated several VMs and ran the weather simulations. Every possibility was considered to allow discarding worst-cases scenarios and tune the remaining. Firewall and proxies were also configured to avoid network issues. Figure 6.9 illustrates the analyzed scenarios.

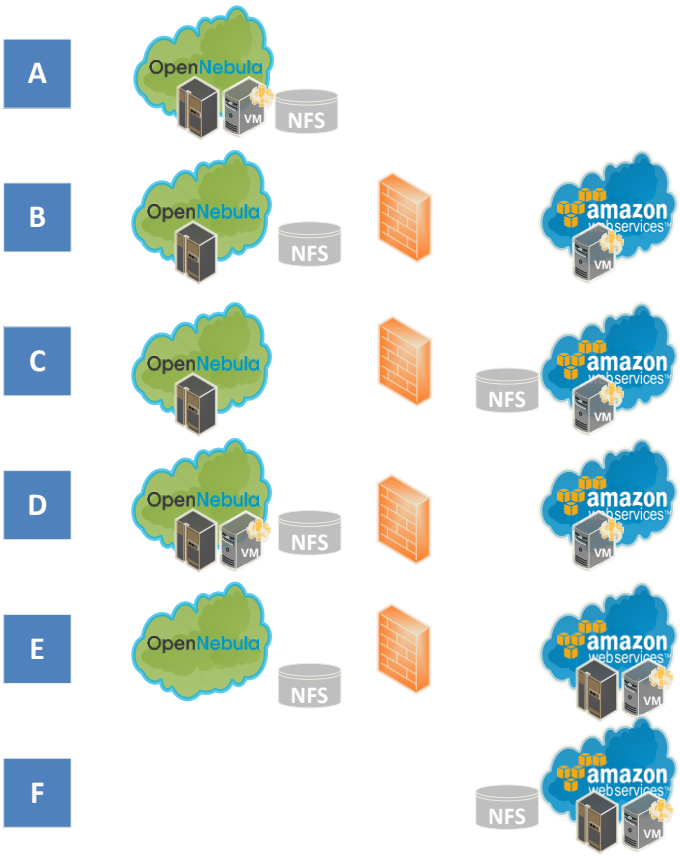

Figure 6.9 - Tested scenarios

The scenarios were set to optimize not only the servers' geographic location but also the storage, a very important part of the system to test. On our experiments, we successively changed the location from on-premises resources to outside resources to foresee all possible results.

6.2.5 Cloud-context issues

After setting the possible scenarios, we ran the forecast simulations for a single period of a day (6h) to maintain equal data. To experiment a wide range of options and variables, these six scenarios were built changing master, slave and disk locations. Starting the scenario experiments, the first simulations were run with a single virtual server (scenario A), which acted as both master and slave for simplification purposes and storage via NFS.

After the starting configuration, some concerns have popped up. The virtual execution time was 7h19m18s, almost 1.5 times bigger than in the original cluster: 4h56m30s, as shown on Table 6.2.

Table 6.2 - Preliminary results

Scenario	Virtualized	Hyper-threading	NFS	Description	Elapsed Time
A	no	no	UA	Original configuration	4:56:30
A	yes	no	UA	Original configuration	5:56:20
A	yes	no	UA	Untuned VM run	7:19:18
A	yes	no	UA	VM w/ Disk tuning	5:29:19
A	yes	no	UA	VM w/ CPU+Disk tuning	5:06:24
F	yes	yes	AWS	VM w/ CPU+Disk tuning	5:17:22

After checking some monitoring values, the problems were found. The first bottleneck found was the I/O reading and writing between the head node and the slave nodes. To optimize it, we performed an I/O test, with iOzone [140]. iOzone is a NFS program that takes various file operations as basic workloads to test the I/O performance of filesystems and adopts diverse testing. Block operations include read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read/write and many others. We managed to change the default CentOS NFS block size from 8 KB (both read and write) to the optimized values gathered from the tests, 65536 KB. Figure 6.10 shows the write test results, where speed is measured in megabytes per second. Bigger differences can reach 30 MB/s but the top 6 differs about 4-5 MB/s, which is still enough for slowing down performance.

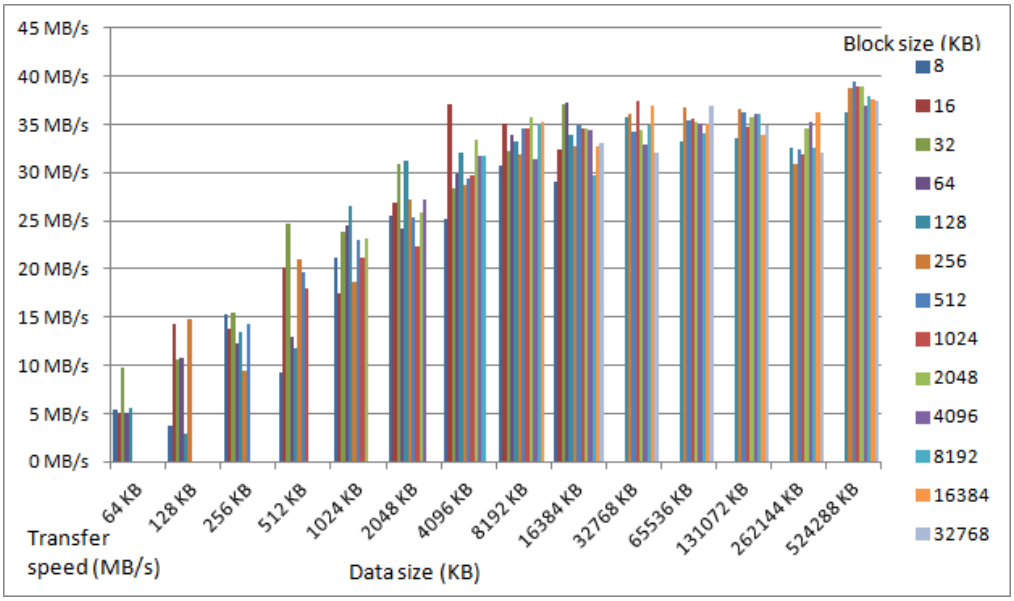

Figure 6.10 - iOzone results (Write Report)

After setting the new block size, the results improved 25%, but this was still a modest performance with a time of 5h29m19s. Suspecting of virtualization problems, we added some raw data with CPU features to the OpenNebula model template. These flags (rdtscp, pdpe1gb, dca, tm2, etc.) allow the VM to have more control over the real CPU cores to push their capacity, though they are very model-specific. There was an improvement of 7% over the I/O optimization and the whole performance improved about 30%, enough to reach a balanced performance, 5h06m24s. With this elapsed time, we could finally start the VM deployment scenario. We ran the simulations on our sandbox blade with and without virtualization.

Elapsed time with full core usage (12-core) reached 4h56m30s on the physical machine and 5h06m24s on the virtual machine (Scenario A). As expected, in a virtualized environment the performance drops due to hypervisor overheads in about 3.2%, which seems reasonable for our experiments.

Another problem was the hyper-threading enabling, which led to worse results. Hyper-threading (HT) allows the CPU cores to share workloads, which normally can improve parallelization. The original server had HT deactivated, but on a shared Cloud to utilize efficiently the resources we could not afford to disable it. Amazon EC2 also has this processor capability enabled, so it was also more consistent to build a faithful comparison.

6.2.6 Results

When constructing hybrid scenarios, some problems have emerged due to servers' geographic locations. The UA, located in Southern Europe, and the used Amazon servers, located on East USA (the cheapest) had some distance and many hops between them. The latency between nodes varied

between 100 and 200ms, depending on working hours on both sides. The bandwidth was also hour-dependent, with a minimum of 1.0MBps and a maximum of 4.1MBps for downloading from Amazon to UA and between 1.4MBps and 7.6MBps for downloading from UA to Amazon. As shown in this section, this has hampered the results of hybrid scenarios.

After running the simulations, the average results of three runs were gathered on Table 6.3, ordered by execution time. The prices shown take only in consideration the public Cloud usage prices, in-house costs as electricity and maintenance were not included. Regarding the private Cloud cost, there is no additional cost in using the existing campus' servers as they already consume power, cooling and maintenance staff, whether they use 10%, 50% or 100% (though there is a slight difference on intensive computing consuming).

Table 6.3 - Simulation results

Scenario	Headnode	Slavenode	NFS	Cores	CPU Clock	$/h	$ total	Elapsed Time
F	AWS	AWS	AWS	12	2.60	$2.40	$12.00	4:34:31
Orig.	UA	UA	UA	12	2.66	$0.00	$0.00	4:56:30
A	UA	UA	UA	12	2.66	$0.00	$0.00	5:07:22
E	AWS	AWS	UA	12	2.93	$1.30	$7.80	5:36:20
B	UA	AWS	UA	12	2.93	$2.40	$14.00	5:34:44
D	UA	UA + AWS	UA	12	2.66 + 2.93	$2.40 + $0.00	$16.80	5:41:30
C	UA	AWS	AWS	12	2.93	$2.40	$16.80	5:45:10

Every simulation managed by MPI, launched a program thread by CPU core, the equivalent to the legacy system. The selected hardware from Amazon was the closest as we could get to our infrastructure, Intel® Xeon® processors from 2.60 to 2.93GHz. A precise match is very hard to reach due to hardware heterogeneity. Even when CPU clock is nearly the same (2,60 and 2.66GHz), the operations per second may differ because of CPU design/brand. We can also rule out result normalization, because the columns to compare are straightforward: time and money.

6.2.7 Considerations

About the data processing, there are penalties on late results, which have the same effect as providing no results. In fact, the costs in not being able to deliver results to clients would reduce the monthly income and could prejudice future contracts.

For a balanced planning, we took into account the cost of Cloud processing, either private or public, and the acquisition of new hardware, with all the associated costs: maintenance, cooling, power and warranty. A new similar blade, a HP ProLiant BL460c Gen8 E5-2650v2 32GB Server, would cost about $3478 with VAT [141]. We can neglect install and maintenance - the IT center has space and all wiring/cooling in place; HP has a 3-year direct replacement warranty [142]. The blade's additional energy consumption is 0.3KWh when workload is 100% [143], which multiplied by the current 0.12$/KWh (enterprise medium cost in Portugal, assuming an equal probability of failure all day) sums a total of 0.86$ per day and 25.92$ per month.

On the public side there is also the cost of data transfer to and from EC2. Data transfer into Amazon EC2 from Internet is $0.00 using a private IP address and out of EC2 is $0.12/GB, up to 10TB per month [25]. This cost was not included in the previous results section because it was below the 1GB limit (0.7GB), but on an extended use it becomes rather significant – 2.8GB per day.

Table 6.4 shows the data comparison relative to all solutions using a single 24/7 solution. We have 4 daily forecasts (6h), which sum a total of 120 per month, and 1460 per year. The new scenario of new hardware acquisition is also considered on a 3-year basis. Usually hardware has a 3year depreciation, where it will be fully deducted on accounting terms and obsolete on computing capabilities, so we built a comparison with costs for a 3-year length.

Table 6.4 - Detailed costs of observed solutions

Scenario	Usage	Initial cost	Per 6h forecast	Per month	Per year	End of 3rd year
F (Amazon EC2)	100%	$0.00	$12.08	$1377.75	$16762.61	$52915.83
Original	100%	$0.00	$0.00	$0.00	$0.00	$0.00
A	100%	$0.00	$0.00	$0.00	$0.00	$0.00
New hardware	100%	$3478.26	$1.02	$4.07	$122.22	$4461.17
F (Amazon EC2) (1% fails)	1%	$0.00	$12.08	$14.77	$177.24	$531.72
New hardware (1% fails)	1%	$3478.26	$79.25	$96.89	$1162.71	$3488.14
F (Amazon EC2) (5% fails)	5%	$0.00	$12.08	$72.49	$869.85	$2609.55
New hardware (5% fails)	5%	$3478.26	$16.33	$97.96	$1175.58	$3526.73
F (Amazon EC2) (7% fails)	7%	$0.00	$12.08	$98.44	$1181.26	$3543.77
New hardware (7% fails)	7%	$3478.26	$12.08	$98.45	$1181.36	$3544.09
F (Amazon EC2) (10% fails)	10%	$0.00	$12.08	$144.97	$1739.70	$5219.10
New hardware (10% fails)	10%	$3478.26	$8.28	$99.31	$1191.74	$3575.21

We also made calculations for a hybrid solution, using the local infrastructure as a starting point and bursting to the public Cloud on failure occurrences, for instance occupied resources or technical fails. The graphic on Figure 6.11 shows the point of interception between the use of new hardware and resorting to scenario F on failures occurrences on a 3-year basis, about 7% of fails. This means that over this percentage, on the considered period, resorting to a public Cloud solution for shorteage of operation in the local cluster is more expensive than a new hardware acquisition.

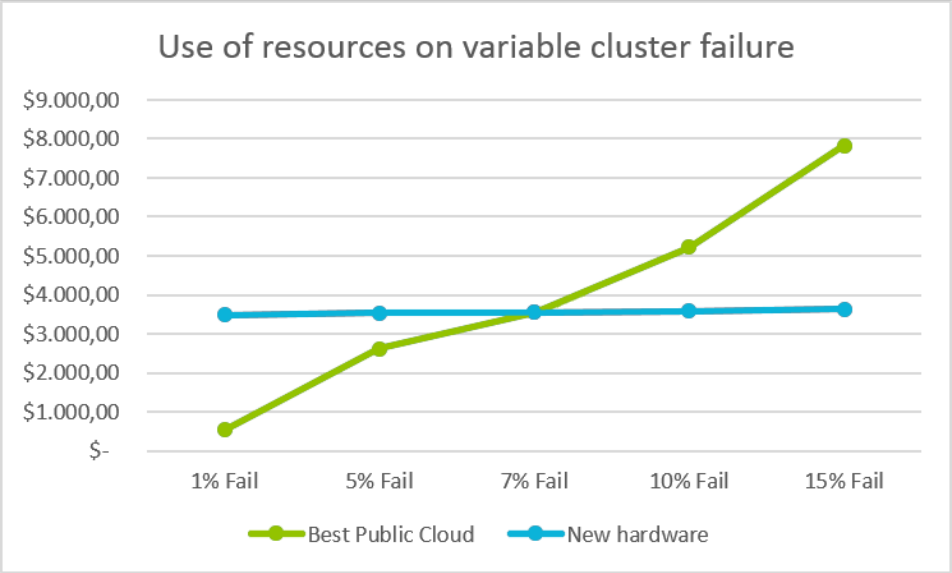

Figure 6.11 - Solutions' cost break even

6.2.8 Results analysis

In [136], authors compare the performance of systems in a virtualized environment and on a physical environment, reporting about 5% CPU loss and 30% for I/O, so we were expecting bigger virtualization overhead losses but the outcome revealed a more narrowed window with non-virtualized/virtualized without HT (about 3.2%) and a largest one with virtualization with HT (about 6.5%). The first hybrid scenarios, E and B, had elapsed times approximately 11% higher than non-virtualized environment, while D and C had outcomes in the 14-15% range. This was a good result for a hybrid scenario, and validates using available on-premise resources, with no additional cost inside the university campus, but also allows assuring simulations on schedule if problems like hardware crash or over demand, with Cloud bursting to public Cloud providers.

The experiments with more VMs were also made, with performance loss around roughly 7% per division. For instance, running 3 VMs with 4 CPU core each is 7% slower than running 2 VMs with 6 cores, which may be acceptable if there are no time constraints and to fulfil campus machine usability – from old to state of the art machines. In summary, there was no gain in dividing worker nodes, neither in time nor in money, when launching VMs should be launched the bigger possible specifications.

The problem with hybrid scenarios was the latency between both the master and slave nodes, but also the latency on NFS, tested on both sides. The latency problem could be solved with a more direct and dedicated connection between Amazon EC2 and UA and with the use of a public Cloud geographically closer, like Amazon Europe in Ireland for instance. Deployments in the public Portuguese universities' network can also work well, since the internal bandwidth situates around 80MBps for download and 10MBps for upload respectively (regardless of the combined sites). In terms of NFS the latency issue was already expected, as authors in [144] also point the same problem, although these times (seconds) can be mitigated within the final results, in a bigger order of magnitude (hours).

Cost-wise, considering just the costs presented in Table 6.3, scenario F (entirely AWS, the best performance time) is second to scenario E by $4.20 per simulation, costs included. However, scenario E is more than 1 hour slower than F. With a better communication link between AWS and UA, we believe that it would be possible to lower both the costs and the processing time, using local and public resources.

A full HPC Cloud solution running on VM is very application dependent. One solution can not fit all datacenters, user requirements and constraints. In our case, the disk block size made an enormous difference due to the GFS data, which came on tens of 100MB files. Another detail to notice is memory, which was not practically used by the algorithm but on the contrary is used thoroughly on many other applications. Note that only CPU is being compared, the forecast application does not rely on RAM. Software, compilers, libraries and kernel tests were not considered as the focus was not application oriented and to provide a Cloud solution directly comparable with the local solution.

While searching for the best and cheapest solution, we could state that a full migration to the public Cloud would be very expensive at a medium-term period. As we could see on Figure 6.11, assigning tasks to public Clouds is economically viable to scenarios up to 7% of local node fails, in our case.

We must also take into account that measuring performance in virtualized environments is complex; there can be changes in running VMs' usage and other services that can influence the entire infrastructure. Authors in [145] offer a whole set of table Cloud providers' benchmarks and yet it is not very easy to state the best within the results due to undisclosed hardware and vast heterogeneity. There isn't only a single variable to compare, a little like CPU comparisons which cannot be only evaluated by their clock speed, which make things harder. Although the migration result and cost can

still be questionable, it is clear that Cloud Computing can provide rapid access to computational resources as and when needed without the need for significant financial outlay [146]. However, a Cloud environment avoids strict capacity planning during execution, and becomes an unpredictable scenario for applications such as the tested weather forecast.

6.2.9 Further work

To optimize this model, the head node must be centralized on a location with the best possible network infrastructure to reduce latency in data flow to the nodes. Also, the network should have priority on a QoS or use separate bandwidth to avoid other network data interference, but this can only be accomplished if all tests are successful and benefits are clear to the structure owner.

As other Portuguese public universities, the UA has its Internet connection dependent on FCCN, a government institute that regulates and manages the infrastructure, which in turn depends on GEANT, the pan-European research and education network that interconnects Europe's National Research and Education Networks (NRENs).

Figure 6.12 shows the major difference of upload and download rates between UA, University of Minho, Amazon and the general Internet. This is a very big bottleneck to hybrid Cloud Computing, intensive applications should not be delayed by network problems.

Based on this network tests, we have requested more bandwidth from UA to Amazon EC2 for further testing and enhancement.

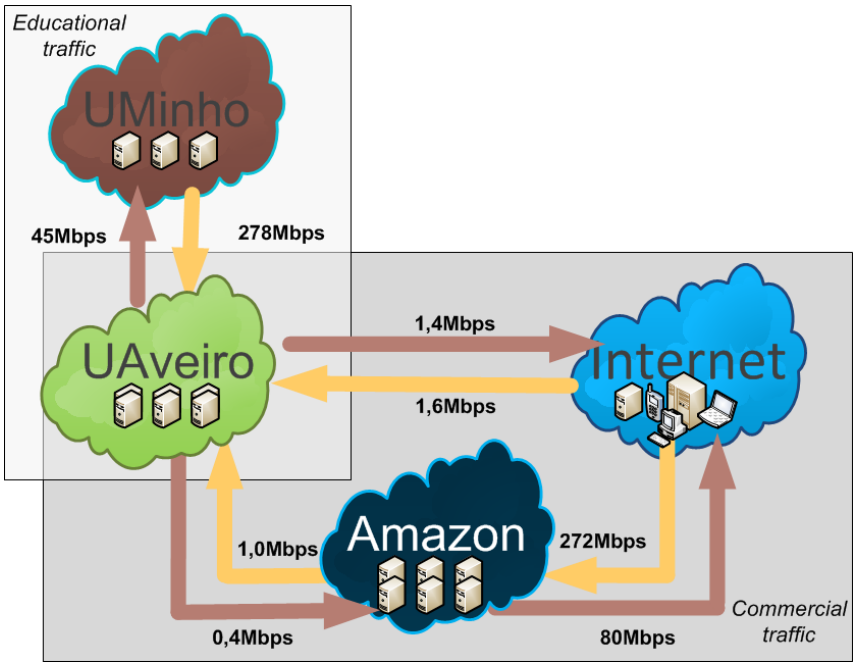

Figure 6.12 - Transfer rates on educational and commercial traffic

There is also room for improvement in terms of shared storage. Although we used the last version of NFS, version 4, there are some solutions that could fit better, like Lustre, GlusterFS and MooseFS, among others, which seem to be a good enhancement for shared storage [147]. One major benefit from running the NFS on Amazon could be the download of the GFS data from US servers; the problem is that the last file is only uploaded and available almost 50 minutes after the first. The batch process itself should be optimized for start processing with the available data and with a higher focus on the wrf.exe task, which occupies the largest amount of processing time. And of course the parallelization process, made in MPI, could be tested with the MapReduce [128].

Considering the simulation costs, public clouds have a very dynamic price range, and it is possible to automatically spot prices that can be used wisely [148], saving budget. Still, it is very hard to predict the users' needs. It is also difficult to preview when these demands overcome the campus computing capacity and there is the need to burst outdoors.

The campus clusters elicitation can be a hard task, but it has already started; only massive adhesion can make the whole project work. However, one must recognize that there is a point where adding more VMs to the processing task will not improve results, especially when the node's hardware specifications are low-end machines.

6.3 CloudCampus

Many UA's departments projects, funded or not, require computational resources to fulfil their tasks. Much of it is bought, used and afterwards left aside or passed on to other projects. The UA IT center also has a cluster infrastructure to let to faculties by request. In addition to this, we have the research labs which have also good resources. The majority of resources is not shared because of its visibility scope, restricted to inside view, but also due to the inherent technical difficulty in sharing.

On the other hand, UA Campus' PC labs have an enormous potential regarding its cardinality. With combined efforts, they can be integrated on a Cloud Computing infrastructure, similarly to their capability of integrating grid-like projects such as Seti@home or Folding@home [102]. The major difference is its capability of virtualization, which allows to run several projects simultaneously, scaling in or out according to the needs. The resource network might also be extended with User-Provided Cloud Computing (UPCC) computers, which can capitalize, on the user's behalf, the underused computational resources on his laptop, PC or small server, while allowing the user to continue using his computer [8].

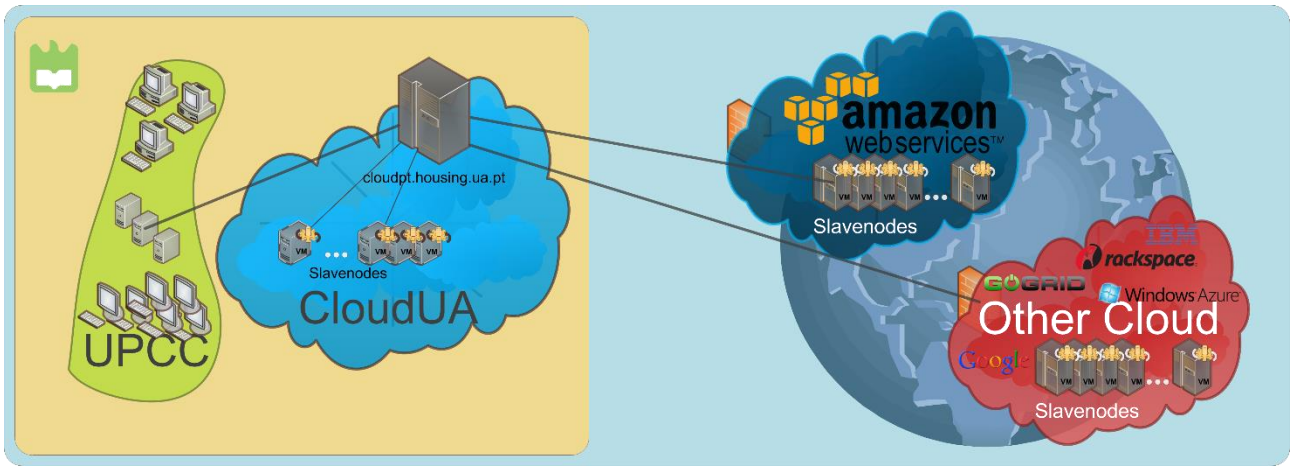

Figure 6.13 - Cloud on premises and external connections

The proposed model, shown on Figure 6.13, can facilitate the resources' commitment and make the most out of their potential. The CloudCampus infrastructure encompasses the campus' hardware, thus providing means for minor processing tasks on PCs. The Cloud bursting to external resources is also an excellent mean to increase capacity when the load overcomes inner resources. The choice of the public Cloud would be delegated to the scheduler, which is able to make calculations of the best prices. Constructing a hybrid Cloud leverages these scenarios seamlessly, by providing a common interface for resource provisioning.

6.3.1 The CloudCampus application

While combining scattered hardware resources for Sky Computing [149] can be quite difficult, combining them within a regulated infrastructure as an university campus can ease the task. Instead

of stating which hardware to lease out, when and for how long, along with its network flow problems, there is always hardware available and a local network whose bandwidth can be customized.

The roadmap moved alongside virtualization, with virtual machine templates and deployments on the private Cloud. At this point, there was consolidation and automation of the processes. Then, it continued with the expansion to the public Cloud, with simulations, network bandwidth tests and automatic deployment to the public Cloud when needed. This assured increased flexibility for peak workloads, the so called Cloud bursting [24]. By stating the availability time, the Cloud management system can schedule activities and make full use of the hardware on campus.

Not every application has the same purpose and computing time frame, so we decided to categorize them. This categorization is used to define quality of service targets among applications. Applications that have requested resources are categorized when registering, so that they can be prioritized not only when resources are low, but also with an adequate time frame for their outcome. As for the case of weather forecast, a late forecast would not be a suitable result. We divided them in three categories, Real-Time, Semi Real-Time and Deferred, as illustrated in Figure 6.14.

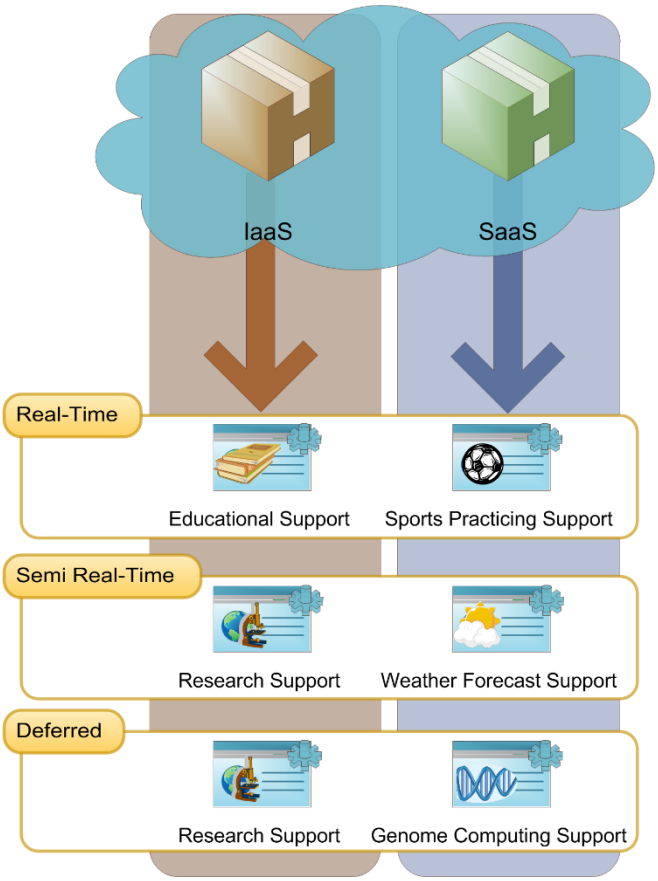

Figure 6.14 - CloudCampus scheduling categories

Real-Time (RT) applications need a fast processing to quickly deliver results; examples include sports statistics and class experiments. Semi Real-Time (SRT) applications enclose computing that is not needed in runtime but have time constraints; for instance weather forecast and class experiments. Deferred (DF) applications define a class of applications that can run and stop whenever the scheduler needs for allowing more important tasks to be run and on the other hand fill up resources at full speed when the slots are empty; examples of such applications are Genomics research and other batch processing tasks.

Figure 6.15 illustrates a typical and estimated use of the CloudCampus' infrastructure. The line graph on top shows the hourly evolution of computing capacity demand, floating as the applications request the capacity; when they overcome the existing resources (sky blue dotted line) CloudCampus scheduler bursts to the public Cloud if defined on template. DF applications are mostly scheduled

for empty slots, which occur typically at night. During the day they have a minimum quota for running, but they are put on hold for as long as there is another application with higher priority. High and low usages are somewhat predictable within classes' time (first classes 9h, lunch time 12h-14h, etc.), so the scheduler lowers the DF quota to let space to other categories.

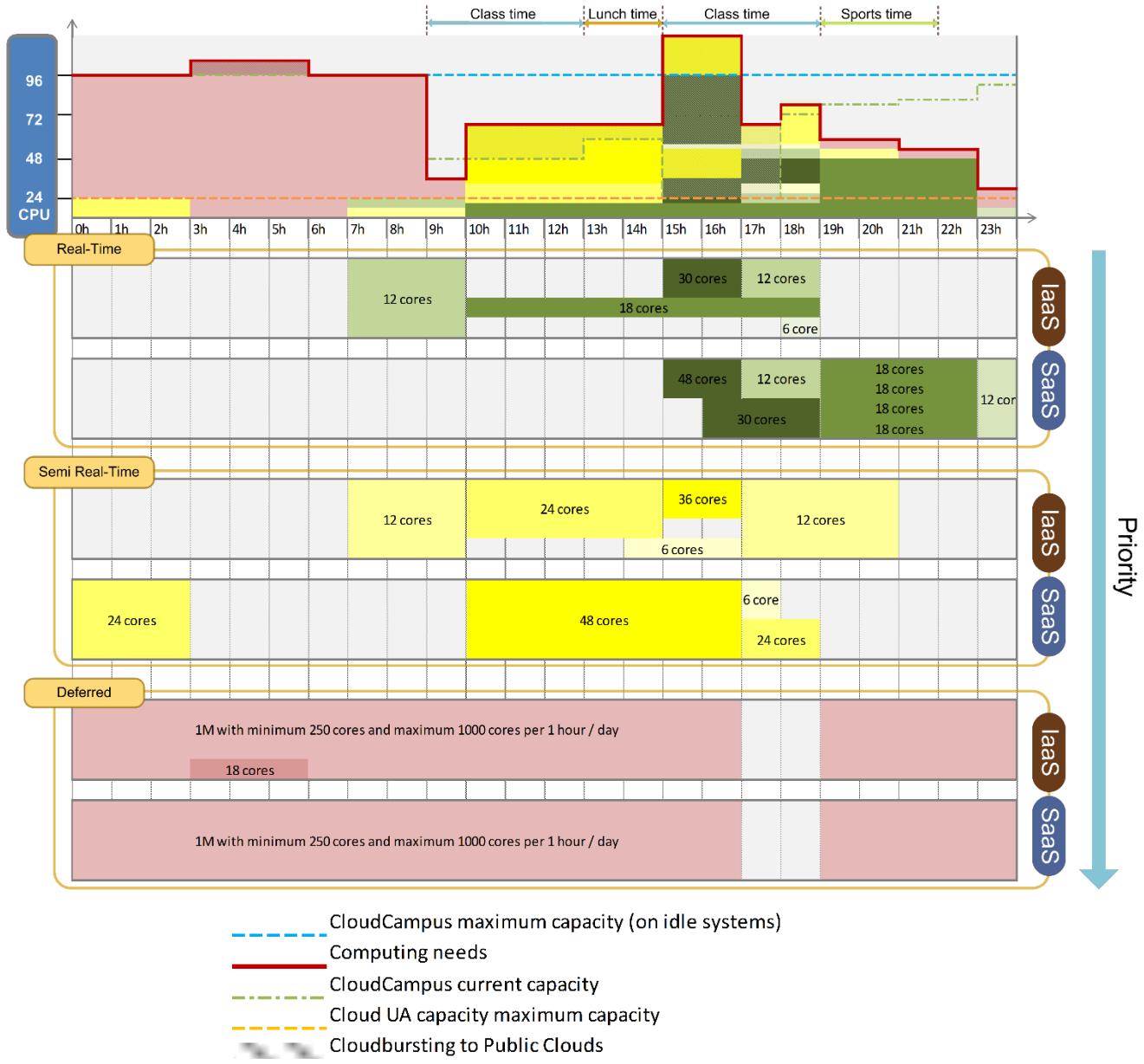

Figure 6.15 - CloudCampus scheduling example

Within each category, the slots are also filled with the highest integer a host can hold, so that smaller applications can be easier to fit on another host along the campus' infrastructure.

6.3.2 Development

Taking advantage of the built Sky Computing infrastructure, we started to adapt the software to the demand. It began on a blade with an Intel X5650@2.67GHz with 12 CPU and 96GB RAM, but with the good acceptance and utilization, quickly we added two more blades to the infrastructure: one more blade with higher specifications – 12 CPU Intel X5675@3.07GHZ with 196GB RAM – and a smaller blade from a finished project – 4 CPU Intel E5420@2.50GHz and 14GB RAM.

The several templates at the user disposal were also a combined effort to satisfy requirements. We added various flavors of Linux distributions but also variations of Windows machines, a novelty at

the time between the KVM and OpenNebula, shared with community [150], [151]. This Windows guest compatibility was developed with Powershell, VB and WMI, which allied to the contextualization mechanism of Opennebula allowed an unattended automatic deployment of Windows VM.
These scripts, on Appendix A1, followed a simple method: after a fresh Windows install, the startup script is registered on Group Policy Manager, which points to a script on CD-ROM. On each VM instantiation, OpenNebula creates a contextualization CD with variables and files described on the VM template. On a new VM instantiation, OpenNebula copies the Windows disk and deploys it under the template hardware specification, Windows boots and runs the CD-ROM script, which configures username, password, IP, gateway, DNS servers, Remote Desktop and so on.

In addition to the updates on OpenNebula, we have also developed a small module to allow more control to the end-user and add HPC jobs, as shown on Figure 6.16. This module displayed on a web form can be a good incentive to current and new users, allowing innumerous possibilities.

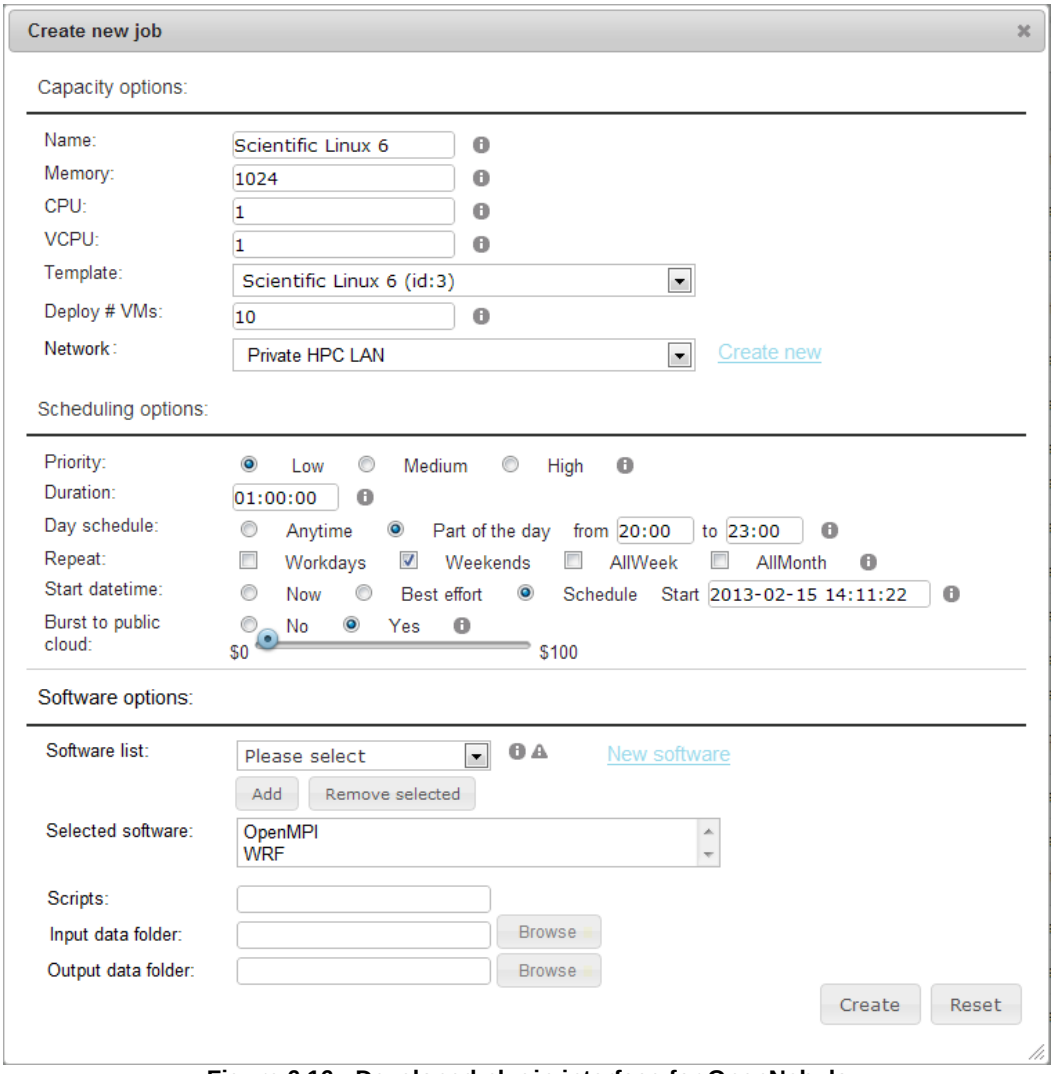

Figure 6.16 - Developed plugin interface for OpenNebula

Some of the existing needs were translated into custom functions in our IaaS. However, with the constant development of the chosen projects, some of them were no longer needed as a new improved feature was set in place. Nevertheless, we implemented some add-ons and contributed to the development of OpenNebula, with debugging, testing and coding.

6.3.3 Adding small nodes

To accomplish the addition of regular personal computers (PC) to the existing Cloud, there were some mandatory requirements. One of them was the CPU capability of virtualization, either AMD

(AMD-V) or Intel (VT-x), the other was a reasonable processing capacity and memory with a minimum of 2,0GHz and 3,0GB respectively. With average overheads of about 5% for CPU [136] and a percentage of memory that is being used by the OS, approximately from 0,5-1,0GB. This was equivalent to a worst case scenario of a 1,9GHz with 2,0GB of RAM machine, although hardware lease can be difficult to define on the Cloud.

The next step was to elicitate the campus' labs to search for fitting hardware. We started with a lab with 9 PCs which had Intel i5 quad-core CPU@3,10GHz with 4,0GB RAM specifications. After enabling the hypervisor on the BIOS, we installed a fresh CentOS6.4 with some mandatory packages: QEMU-KVM, SSH and NFS among the most important. The configuration was also made adding the oneadmin user (global user id across all CloudCampus), editing the firewall, adding a network bridge and mounting the datastore for image sharing (Appendix A5). With all this set, the PC was effectively on the Cloud and a test VM was launched successfully.

Benchmarking

Not every resource on a VM is absolutely granted, which can corrupt results and consequently comparisons. The number of virtual machines usually is greater than the number of cores in the system. To enable an agile sharing of the CPU cores, the hypervisor defines the notion of a virtual CPU (VCPU), where each virtual machine is assigned one or more VCPUs at VM creation time; the hypervisor allows for a flexible mapping from VCPUs to physical cores in the system [152]. A physical CPU core can be allocated exclusively a VCPU or shared among several VMs' VCPUs.
If multiple VCPUs are assigned to a core the hypervisor splits time on the cores between the existing VCPUs.

In our benchmark we measured several performance properties including CPU, memory and network, except for I/O because of its shared nature (NFS). The benchmark tests were made with the Phoronix Test Suite, a comprehensive open source testing software and benchmarking platform available that provides an extensible framework for which new tests can be easily added [153]. The normalized results should give notion of the computing capacity of an aggregated lab room, and also of each PC.

Methods

The best way to compare apples to apples was to create a single VM template for both physical machines (the cloud server and a PC), with 1 CPU (with a one to one relation on CPU-VCPU) and 3GB RAM, almost 1GB was being used by the PC OS. The image used had a pre-configured CentOS 6.3. The specifications are summarized on Figure 6.17.

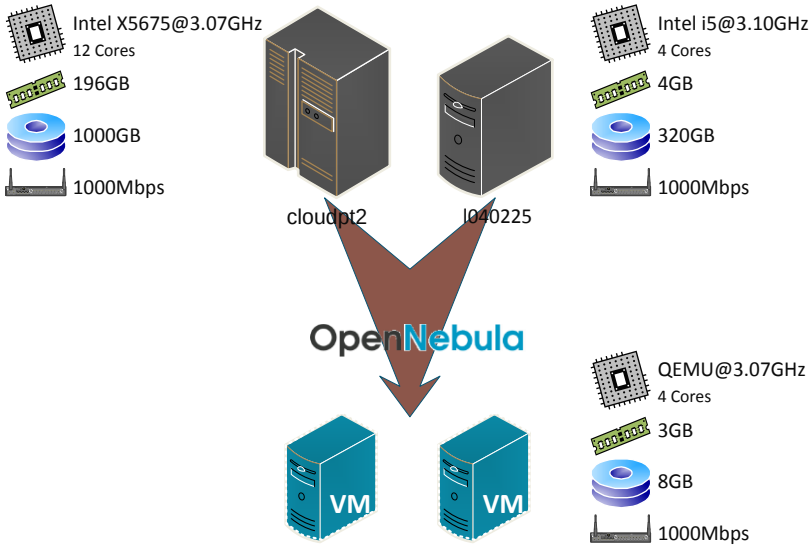

Figure 6.17 - Benchmark machines' specification

With instantiated VMs, we installed the Phoronix software packages and started the tests. Some tests have already pre-determined iterations, but we repeated them at least 50 times to make it more trustworthy.

Selected tests

Benchmark tests can differ even on optimal conditions, and it is no surprise that distinct hardware behaves differently with the same tests. We chose a set of tests that encloses several categories for CPU, the main focused feature, and a single test for network and memory. The diversity of these tests was chosen due to reliability and proof of concept algorithms [154], [155], which enclose several computing categories. As the disk was a shared storage based on our NFS server, it did not influence the other categories results.

Table 6.5 - Benchmark tests details

Test	Target	Description	Type
Pi	CPU	Pi to 8,765,4321 digits	Floating point
C-Ray	CPU	Multi-threaded ray tracer	Floating point
VPX	CPU	VP8/WebM encoder	Floating point
GCrypt	CPU	CAMELLIA 256-ECB cipher	Integer
x264	CPU	H.264/AVC encoder	Integer
RamSpeed	Memory	Scale, average, copy, add, triad	Floating point/integer
Network-loopback	Network	TCP performance	n.a.

Table 6.5 shows the selected benchmark tests. For CPU we chose 3 floating pont tests, the most used CPU branch and 2 integer tests. Including a big iteration cycle (Pi), a multi-core aware test (C-Ray), a VP8 video encoder (VPX), a decoding cipher (GCrypt) and an H.264 video encoder (x264) constituted a wide weighting benchmark.

Results

After the installation of Phoronix Test Suite on each VM, one for our server cloudpt2 and one for our lab PC, the tests produced the results on the tables and graphs (Appendix E).
The corresponding graphics show easily comparable data. The summary is illustrated on Table 6.6 and Figure 6.18. Considering the cloudpt2 VM as the reference, the Difference column holds the information regarding the percentual net diference of performance for the PC VM.

Table 6.6 - Benchmark results

Test	cloudpt2 HP Proliant BL460c	Std. Dev.	lab HP Compaq 8200	Std. Dev.	Difference
CPU Pi	4,23 s	11%	5,01 s	14%	-18%
CPU C-Ray	65,96 s	2%	68,99 s	2%	-5%
CPU VPX	18,45 FPS	1%	22,77 FPS	1%	23%
CPU GCrypt	2,64 s	2%	2,87 s	1%	-8%
CPU x264	46,45 FPS	1%	54,70 FPS	1%	18%
RAM	10762,80 MB/s	8%	6456,97 MB/s	6%	-27%
Network	21,78 s	4%	17,80 s	4%	18%

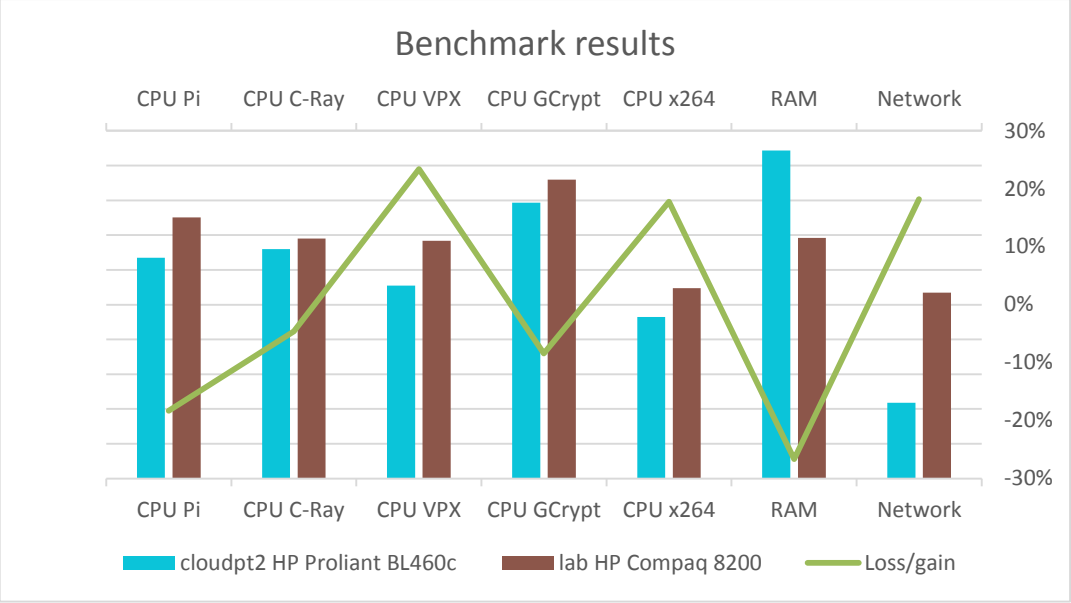

Figure 6.18 - Benchmark results

Though there are very different measures on this benchmark, the results were normalized to appear together in the graphic; they are only comparable between results of the same test. Note the loss/gain ratio that is calculated from the PC/server division, which leads to some positive (better performance on server) and negative results (better performance on PC).

On a closer analysis, the standard deviation was not considerably high. Still, we regularized the results to clear some possible erroneous results, cropping the 5% of best and 5% worst calculus. The subsequent table illustrates the final results.

Table 6.7 - Normalized benchmark results

Test	cloudpt2 HP Proliant BL460c	lab HP Compaq 8200	Loss/gain
CPU Pi	9405,35 s	6920,78	-18%
CPU C-Ray	4,22 s	5,01	-5%
CPU VPX	65,87 FPS	68,99	23%
CPU GCrypt	18,47 s	22,77	-8%
CPU x264	2,64 FPS	2,87	14%
RAM	47,24 MB/s	54,70	-29%
Network	21,76	17,81	19%

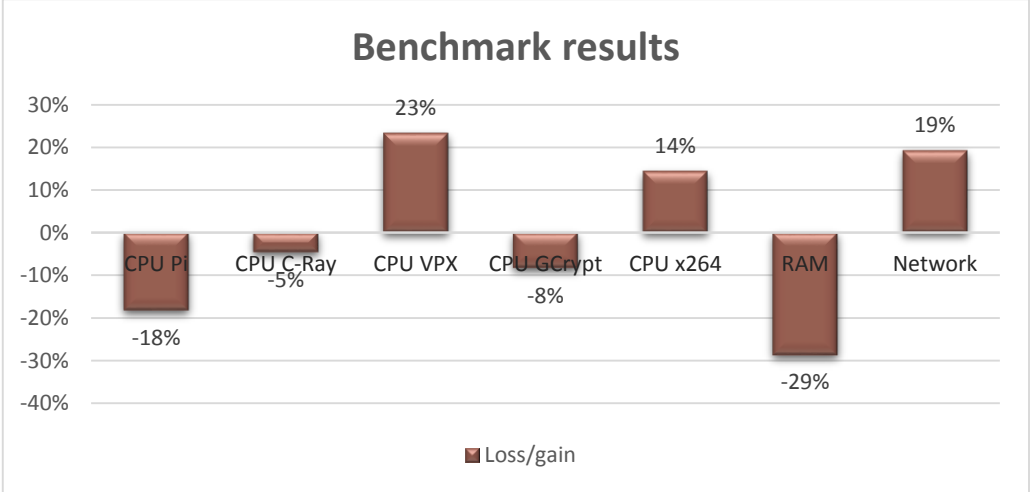

Figure 6.19 - Normalized benchmark results

Also, we took into account that the server's VM was the reference value and the resulting percentage loss/gain is related to it.

Result analysis

Cloud performance is very hard to measure correctly because of its abstraction and multitenancy. Some hardware may be shared under some VMs or not, which deeply affects the output data. Also performance is workload dependent, graphic applications use different CPU resources than encoding algorithms and even some applications can be faster virtualized than in barebone hardware [156].

A first look over the results show that RAM was largely better on the server benchmark, thanks to its improved server specification rather than clock speed - 1067 MHz DDR3 FB DIMM (Fully Buffered Dual In-line Memory Module) ECC against a stock 1333 MHz DDR3 SDRAM. The second strong note is the network loss, easily explained with a crowded server against an empty lab PC, although three more test repeats did not significantly change the values.

As for the CPU results, some tests were favorable to the server and some to the PC. Inspecting the CPU's specifications from Intel [157], [158], we detected features that may explain the obtained data. The i5 CPU has a specific graphics co-processor which justifies the better performance on FPS benchmarks; the Xeon CPU has intrinsic bigger memory bandwidth and a QuickPath Interconect (QPI), the way the CPU communicates with memory, of 6.4GT/s against 5.0GT/s.

Metrics

In order to present a faithful comparison between the obtained data, it is imperative to find similar metrics. A simple approach could use combined results using a mean or sum of their values. Nonetheless, that approach would be very imprecise because all the benchmarks are different and extrapolate different computing features.

One possible approach is to give weights to each test, stating their importance on the workload, and calculate the sum of each test's result. The Standard Performance Evaluation Corporation (SPEC) benchmarks are industry-standard benchmarks, designed to compare servers' performance [155]. The widely used SPEC benchmarks are cross-platform allowing to properly compare distinct architectures. According to the HP Proliant BL460c system stats [159], the memory and network also have a share of importance, thus reducing the CPU to 70%. For evaluating CPU's importance, we followed SPEC's pattern – tests with a 40% division for integer and 60% for floating point.

Equation 6.1 – Tests' relative weight

$$\frac{\mathbf{70\%}\ \boldsymbol{CPU}}{\mathbf{5}\ \boldsymbol{tests}} = \frac{\mathbf{14\%}\ \boldsymbol{CPU}}{\mathbf{1}\ \boldsymbol{test}}$$

$$\mathbf{60\%}\ \boldsymbol{floating\ point\ tests} \times \mathbf{70\%}\ \boldsymbol{CPU} = \mathbf{0,42\%}$$
$$\mathbf{40\%}\ \boldsymbol{integer\ tests} \times \mathbf{70\%}\ \boldsymbol{CPU} = \mathbf{0,28\%}$$

As we can see on Equation 6.1, 70% divided by 5 CPU tests equals 14%, the set of tests is proportional to its importance: 14% x 3 = 42% which represent 60% of the CPU and 14% x 2 = 28% represent 40% of the CPU weight. Table 6.8 shows the summary of the obtained results.

Table 6.8 – Systems' comparison

Test	Target	Type	Weight	cloudpt2 HP Proliant BL460c	lab HP Compaq 8200
Pi	CPU	Floating point	0,14	2,542	
C-Ray	CPU	Floating point	0,14	0,644	
VPX	CPU	Floating point	0,14		3,263
GCrypt	CPU	Integer	0,14	1,157	
x264	CPU	Integer	0,14		2,021
RamSpeed	Memory	Floating point/integer	0,20	5,716	
Network-loopback	Network	n.a.	0,10		1,922
		TOTAL	1,00	10,059	7,206

Multiplying the percentage scores from the summary Table 6.6, only counting the positive values, we get the final scores. In short, the lab PC VM is approximately 72% of the server machine VM. To equal a similar 12-core server there should be almost 6 lab PCs equivalent to the tested ones, with 24 cores.

Aggregating resources

The following stage was to combine the PC's resources, incrementally from a lab, a set of labs, and then the full department. To accomplish this goal, without interfering with the main purpose of the labs – classes – we had three solutions. The first one was to make a ghost image of the VM used on the previous benchmark and install it across all PCs, and access it through the GRUB boot menu; the second one was a little more invasive but more practical: an additional startup script to the already installed Unix OS; the third was to install the OS on a USB flashdrive. We chose the second option, since this would mean that the classes could continue during the external usage of the PC and we could manage it remotely.

The existing Unix OS on the PCs was the Ubuntu 12.04 Server 32-bit. We developed a script to allow initial installation and configuration of the following:

- network configuration (add bridge and DHCP)
- hostname (all PCs have an automatic DHCP name like `l040225-ws12.clients.ua.pt` where 04 is the university's department, 02 the floor, 25 the classroom and 12 the network)
- passwordless SSH to/from CloudCampus main server (to allow using hypervisor seamlessly)
- required libraries (Libvirt, QEMU-KVM, etc.)
- firewall configuration (SELinux, iptables updated)
- user configuration (user, groups and permissions)
- shared folder (mount NFS folder for Cloud image deployment)

The script `installUbuntuUA.sh`, on Appendix A5, was executed by the existing regular update batch. It installed also a second script `autoubuntu`, a batch able to start and stop the service, enabling itself on OpenNebula on startup and disabling itself again on shutdown/reboot.

Having 7 other similar class labs, each one with 9 PCs at least gave us a powerful capacity of almost 190 CPU cores, equivalent to 136 Cloud CPUs. Nevertheless, given its temporary and volatile nature, they provide a meaningful hardware resource on two situations: when existing premises are not enough and also mitigating the existing Cloud servers' effort when redirecting smaller tasks to these particular machines. A next step would be extending this virtualization possibility to all lab PCs, without constraining their main purpose.

6.3.4 Experimenting HPC

The following step was to test a real application on the small node infrastructure. For this experiment, we used the weather forecast application.

Before the experience, we tested the SPEC benchmarks: one of the major concerns was the installed Ubuntu 12.04 architecture: 32-bit OS over a 64-bit processor. The results confirmed the suspicions; the performance was almost half of the 64-bit tests executed with CentOS 6.4. As changing the OS was impracticable, either in time or academically, we opted to build a live USB pendrive with the applications installed and configured.

After an USB install of Ubuntu 12.04 64-bit, we used the scripts to install all the required software and to configure settings, like IP addresses and OpenNebula hosting. We cloned the USB to another 8 pendrives and booted from USB on all 9 machines, with successful attachment to the CloudCampus resources. On OpenNebula, we instantiated the template of a VM per host, used a host to be the headnode, updated the slavenodes list on the master and ran the forecast application. The results are presented on Table 6.9.

Table 6.9 - Small nodes forecast simulations

# VMs	Cores	CPU	SIM_TIME=3	SIM_TIME=18	SIM_TIME=24	SIM_TIME=180
2	8	2	00:15:15	01:31:30	02:02:00	15:15:00
3	12	3	00:12:02	01:11:25	01:36:16	12:02:00
4	16	4	00:09:54	00:58:21	01:19:12	09:54:00
5	20	5	00:10:01	00:58:53	01:20:08	10:01:00
6	24	6	00:09:53	00:58:26	01:19:04	09:53:00
7	28	7	00:10:03	00:58:41	01:14:57	09:22:07
8	32	8	00:10:22	00:59:59	01:22:56	10:22:00
9	36	9	00:10:16	01:05:21	01:27:08	10:53:30

As we can see, the results show poor performance when comparing the best result with the best on-premises result (4h56m30). However, these results result on several conclusions. First, having more CPUs does not always mean better results; secondly synthetic benchmarks may not be the best way to estimate application performance; and third, while these results do not fit to weather forecast, other applications as SRT and DF can apply on our case.

Consumption

Another angle tested on these experiments was power consumption. Back in paragraph 6.2, the estimation costs were made without counting the maintenance or power consumption, with the justification of hardware was already deployed, maintained and online. The results on Table 6.10 show the details of a PC, with full usage, on the tested lab.

Table 6.10 - PC consumption on forecast simulation

Description	Time	Power	%
ungrib	00:00:06	76.00 W	0.0002
metgrid	00:00:25	79.00 W	0.0007
real	00:00:40	78.00 W	0.0012
wrf	09:20:56	86.00 W	0.9979
total	9:22:07	85.98 W	1.0000

On Table 6.10 we can see the several phases of the forecast, with different execution times and power. The considerated cost was 0.12$/KWh, an average of the enterprise tariff calculated with an exchange rate of 1,38€/$ in July 2014. The average of the tests was 85.98W, which for 7 VMs on 9h22m07s equal approximately 0.68$, as we can see on Equation 6.2.

Equation 6.2 - Simulation cost calculation

$$85.98W \times 7 = 601.885\ W\ = 0.601885\ KW$$
$$0.601885KW\ \times \$0.12\ = \frac{\$0.072262}{h}$$
$$\$0.072262\ \times 9.36h\ = \$0.676$$

As for the standby consumption, the measured values had an average of 22W, totalizing $0.17 on 9h22h07s of standby, as shown on Equation 6.3.

Equation 6.3 – Standby cost calculation

$$22.00W\ \times 7\ = 154.000W = 0.154\ KW$$
$$0.154000KW\ \times \$0.12\ = \frac{\$0.018480}{h}$$
$$\$0.018480\ \times\ 9.36h\ = \$0.173$$

Combining the results, the difference of the use of the lab PCs is about $0.50 in a 9h job, with an additional energy consumption of 0.448 KW. The HP BL460c spends $0.14 for each simulation, consuming 0.299 KW; when standing-by it consumes $0.05 and 0.102KW.

6.3.5 Experimenting other applications

Regarding the volatile nature of the campus labs, there are more suitable applications. For instance, a frontend machine without computing capabilities able to have backend workers, which would do the heavy work and would be available dynamically. A solution with webservices would fit our unpredictability on resources and take advantage of the existing hardware at the same time. Webservices can coexist on the campus and provide a meaningful service to users. Flexibility and versatility, two of the best CloudCampus' infrastructure advantages, can fulfill the users this way.

When referring webservices, an integrated model that can be applied to our infrastructure is the Universal Description Discovery and Integration (UDDI) model. UDDI is a protocol that defines a standard method for publishing and discovering the network-based software components of a SOA [160]. Its architecture is depicted on Figure 6.20.

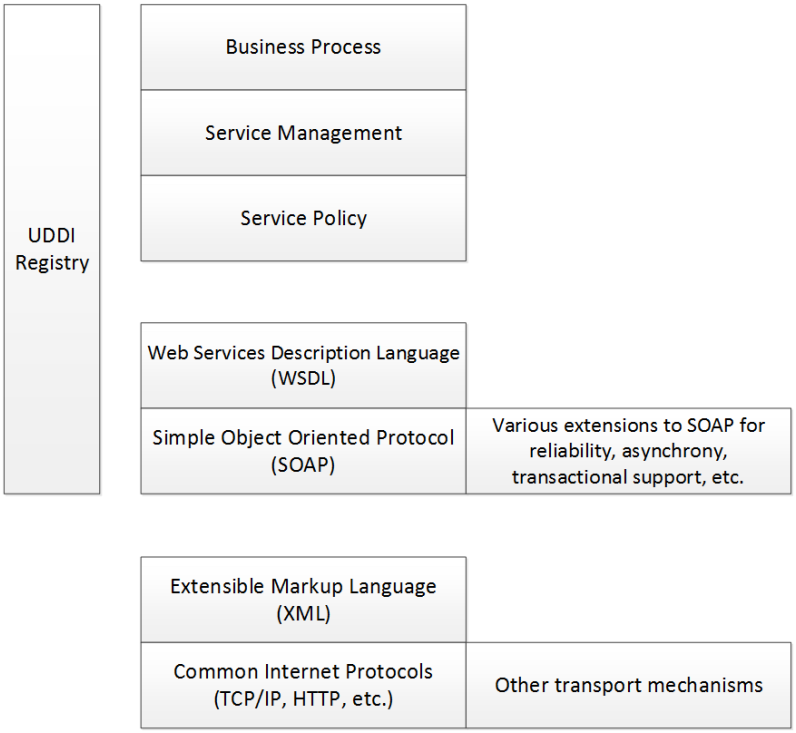

Figure 6.20 - UDDI architecture [160]

Basically, this model works as an archive with mechanisms to register and locate web service applications, defining how the services can be accessed and interacted. The main objective of this experiment was to implement VMs on volatile nodes (PC) running webservices and observe their behavior on client demand. Figure 6.21 iIllustrates the UDDI workflow.

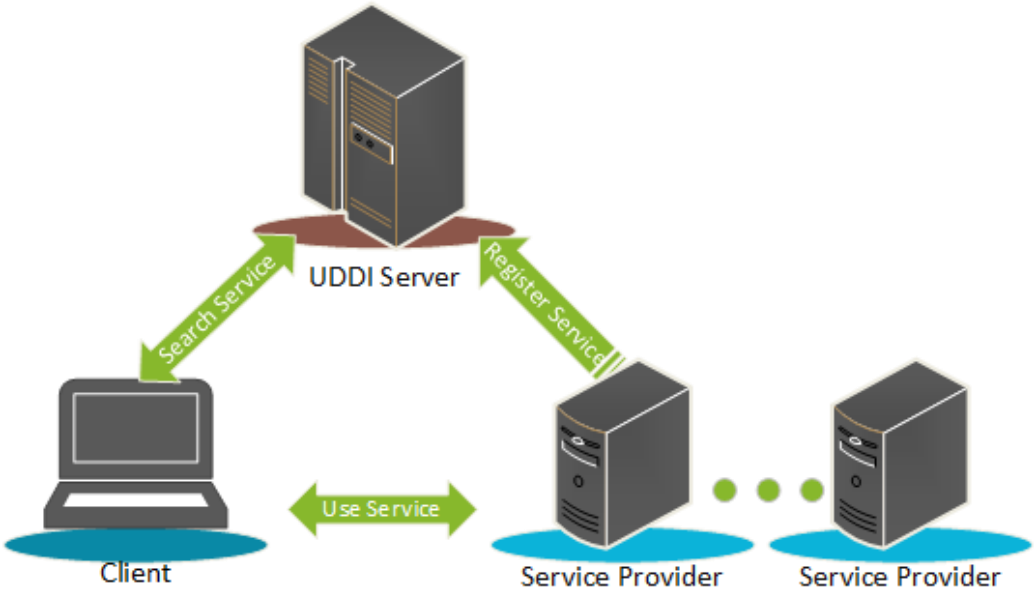

Figure 6.21 - UDDI implementation model

As a proof of concept, we implemented this on CloudCampus, using the same lab classroom with 9 PCs.

Deployment details

To test the whole system we needed:

- An UDDI server, public and able to deal with registrations/requests;
- A webservice provider with an accesible webservice, capable of interact with the UDDI server;
- A webservice consumer, to inquiry the UDDI server for services and consume it from the provider.

The UDDI server was implemented on a CentOS 6.5 with jUDDI, a Java open-source software [161]. This server had a web UI, but most importantly had an API which we could use to programmatically register/deregister services. Figure 6.22 shows the UI with the available Businesses; CloudCampus had our webservices registered.

The webservice provider ran a Tomcat servlet container, with a sample webservice containing a set of three operations: print welcome message, sum and multiply. This machine was modified to run the Java API both on boot and reboot/shutdown, registering/deregistering on the UDDI server automatically. This allowed the webservice access to be used only when the machine was alive. This provider was converted in a VM template to allow replication on the CloudCampus infrastructure, with automatic network/hostname deployment. The API code and boot scripts are available on Appendix A6.

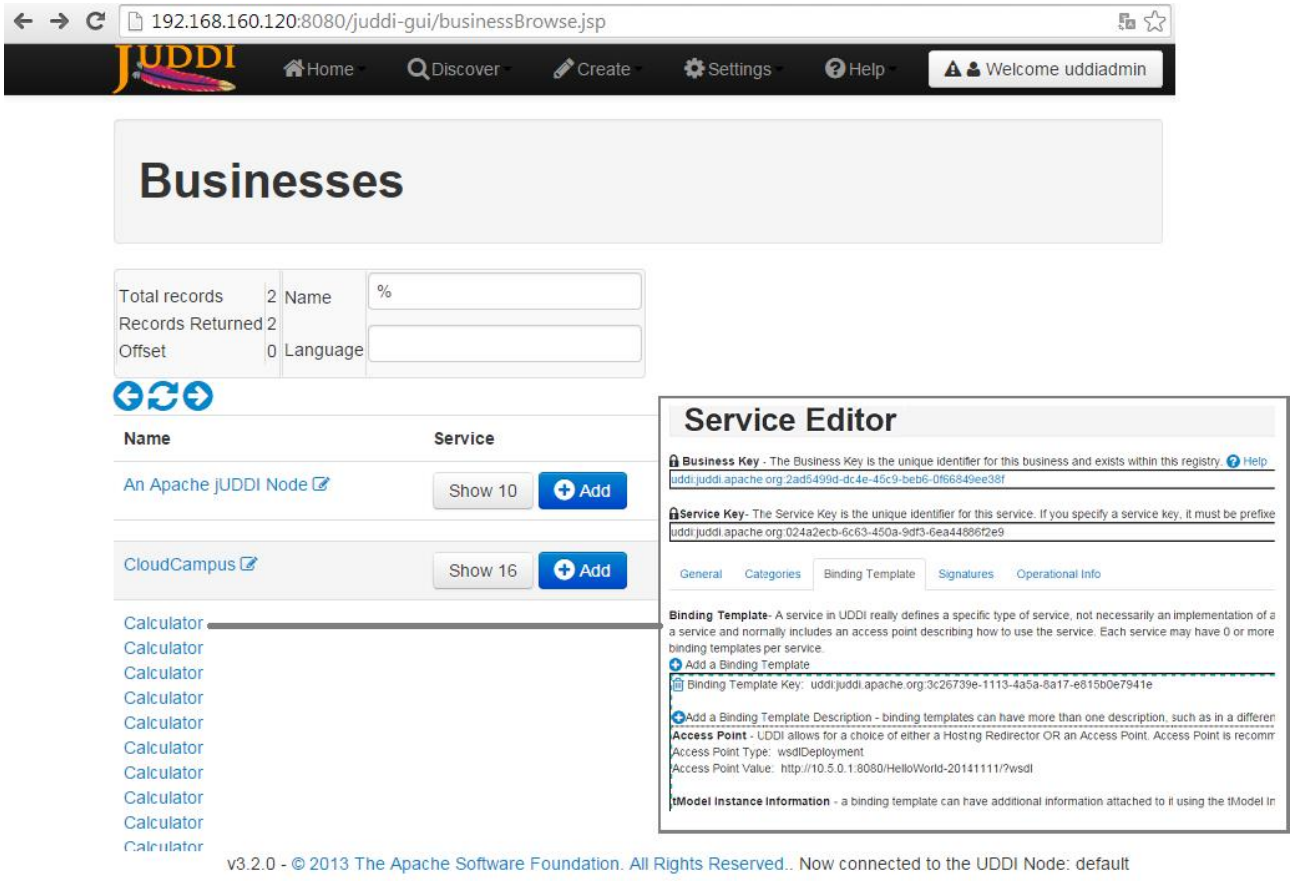

Figure 6.22 - jUDDI server

The client was a simple addon on Mozilla Fireox browser, the SOA Client [162]. It allowed inquiring for webservices on the UDDI server and using them afterwards, as illustrated on Figure 6.23. It also allowed repeating the request multiple times to test reachability.

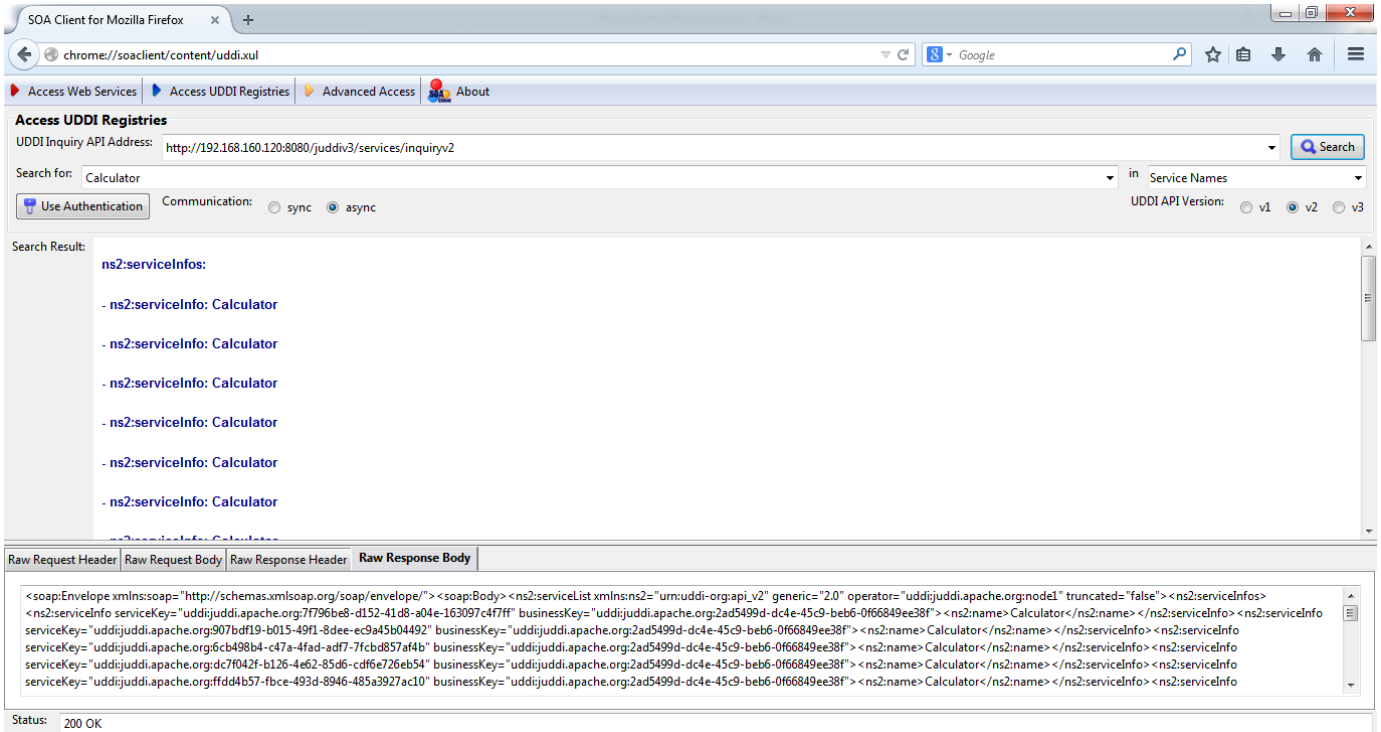

Figure 6.23 - SOAClient addon on Firefox

After all configuration done, we connected the lab PCs to our Cloud infrastructure and launched the VMs. We instantiated 2 VMs per PC on the first run and then 4, totalizing 32 VM. Figure 6.24 depicts the VM monitoring in the middle of the experiment.

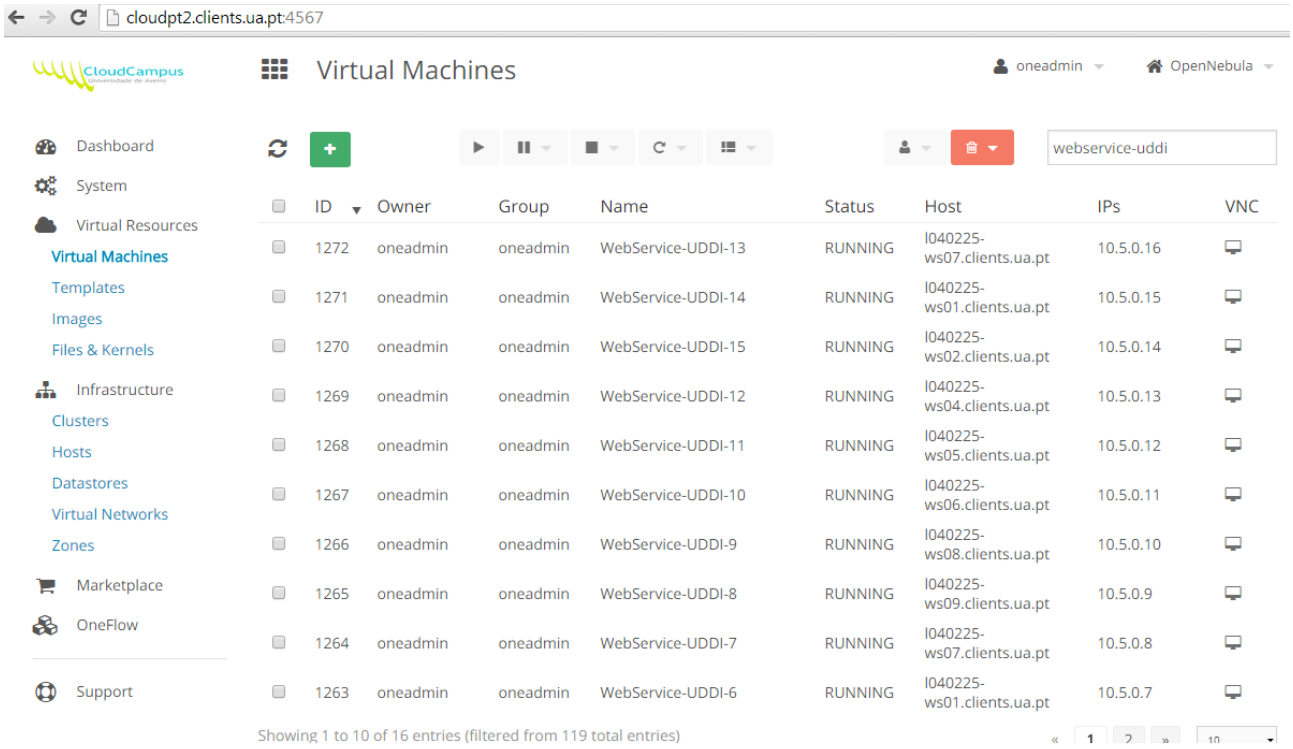

Figure 6.24 - CloudCampus screenshot

We were able to successfully run this concept with an UDDI Java based server on a distributed web environment. During the tests, we continuously instantiated and deleted VMs to simulate the PCs floating availability. This variation between 2 and 32 VMs had no effect on the webservice availability, the webservice always appeared and its functionality was always assured with no more than 3 seconds of delay. This Cloud environment can be scaled up and down according to campus availability, thus supporting a flexible way to provide webservices.

6.4 Application enhancements

The CloudCampus application is an infrastructure built with the presented Sky Computing architecture, even though in the last iteration many components have been replaced which resulted on the product of a modified open source IaaS – OpenNebula – with some addons. Some of them are small tweaks and have been included on OpenNebula development, others are more relevant to specific UA's requirements. The following sections describe some of the developed add-ons.

6.4.1 Federated authentication

Authentication is a process of determining whether a user is who he says he is. This process is achieved by using some data, like a password, a token or a biometric print. Authenticating implies risk evaluation, more important systems require a stronger method that ensures and guarantees safety of sensitive data.

In UA, in association with FCCN, the general user authentication is made with Shibboleth with a unique user (UU) [163]. Shibboleth is a SAML implementation of a two entities concept: an Identity Provider (IdP), the sender, and a Service Provider (SP), which has a validation role [164]. Each user has a single registration with his home organization, which manages his credentials and permissions;

alongside with the authentication process some user-related data can also be requested, as its role on the organization.

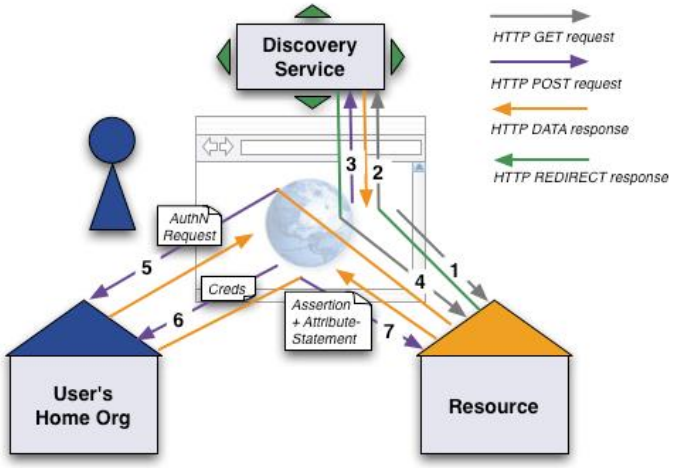

Figure 6.25 - Shibboleth workflow [164]

Figure 6.25 shows the workflow [164] for Shibboleth authentication from the HTTP protocol:

1. User accesses the Resource
2. Service Provider issues Authentication Request
3. User Authenticated at Identity Provider
4. Identity Provider issues Authentication Response
5. Service Provider checks Authentication Response
6. Resource returns Content

In our case, these roles (as student, teacher and other staff) are an important asset when distributing resources within the CloudCampus. There are also subsets of these roles, as a student's class or course, allowing correctly configuring resource quotas. Alongside the user authentication on the web portal Sunstone, which redirects to a proxy to use the IdP, we separate users' permissions and groups. After a successful authorization and identification, users are bound to their respective quota. CloudCampus' scheduler ensures each user is limited to its defined boundaries, thus checking them before launching requests.

6.4.2 Scheduler

The CloudCampus scheduler component is responsible for analyzing incoming requests and designating their destination. If the request has all the correct parameters it will be allocated to a host, if incorrect reports failure. OpenNebula's architecture defines this module as a separate process that can be started independently and the scheduling framework is designed in a generic way, so it is highly modifiable and can be easily replaced by third-party developments [43], an added-value of being open-source.

The scheduler module works in three steps:

1. The hosts that do not have enough resources to run the VM and do not meet the VM requirements are discarded;
2. Using the information gathered by the monitor drivers, a list of hosts is provided and the "RANK" expression is evaluated;
3. The resources with a higher rank are used first to allocate VMs.

There are also three main policies available:

1. The packing policy targets the most used server minimizing the number of clusters in use;
2. The striping policy spreads the VMs to maximize resources in a server;
3. The load-aware policy uses the server with less load first, maximizing individual resources.

Taking this match-making scheduler as a basis, we altered it to more properly fit our infrastructure needs. Firstly we introduced the priority rank for time prioritization, as described on section 6.3, with Real-Time, Semi Real-Time and Deferred applications. Secondly, we used a combined policy with the striping policy, which allowed us to use the small lab PCs. Finally, we introduced the possibility of bursting to the public Cloud, only available for super users because of the additional costs. For such objectives, we modified the `SchedulerTemplate.cc` file, adding the proper code (Appendix B1). The whole process is transparent to the user, simply using the templates takes advantage of this modifications.

6.4.3 Monitoring

OpenNebula encloses monitoring mechanisms, implemented by the Information Manager with so called drivers. When starting OpenNebula, each driver is specific for a hypervisor and it is loaded like a daemon. The KVM driver, for instance, monitors with a pre-defined period CPU, memory and hostname. This is relative to the host; when it comes to VM monitoring, the driver relies on the qemu-kvm process to detail each machine's statistics, which is not very faithful or particularly updated regularly. When researching for monitoring, both Ganglia and Nagios appeared to be a good solution. Ganglia was already supported in OpenNebula but it showed some limitations, i.e. in alerts and reports, so we moved on to Nagios.

The Nagios software was firstly installed on a sandbox VM inside CloudCampus. The main concerns were to detect issues at the first sign of a problem, report them to alert administrators and monitor the entire infrastructure VMs and processes. The software monitoring includes system metrics, applications, services, network protocols, servers, and network infrastructure.

Alerts can be delivered in several ways like email, SMS, or custom script; at the moment it is being delivered as email. We implemented probes to monitor several data, but one of the most used is the storage probe, because the NFS shared data lies on 1TB disk. On the other hand, using qcow images is very fast at the beginning, but as the images can grow, the disk limit sometimes is often reached.

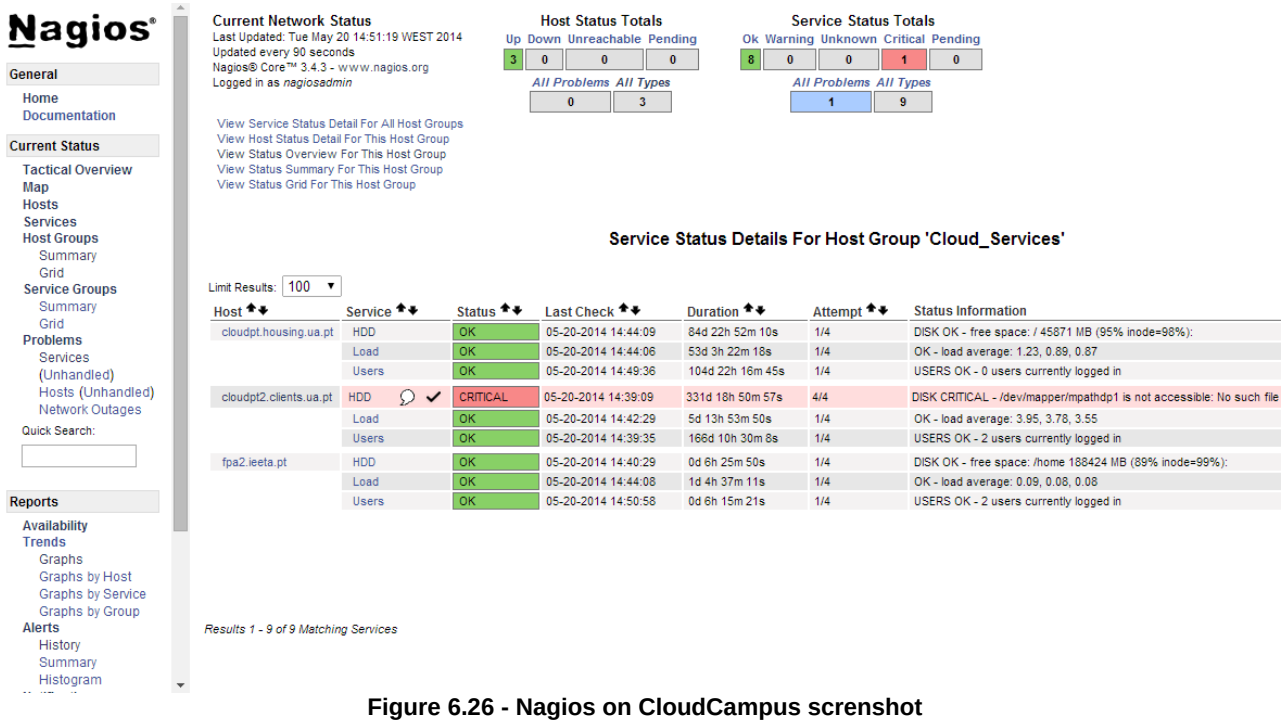

Figure 6.26 - Nagios on CloudCampus screnshot

The main added-value is that Nagios sends alerts when critical infrastructure components fail and recover, CloudCampus is not a public provider with 99,99% of availability, but it should be consistent and reliable to essential research tasks. As soon as the CloudCampus improvement plans are done,

(i.e. storage is migrated to a dedicated storage facility) so will Nagios move to a physical machine to avoid the dependency of its own targeting infrastructure.

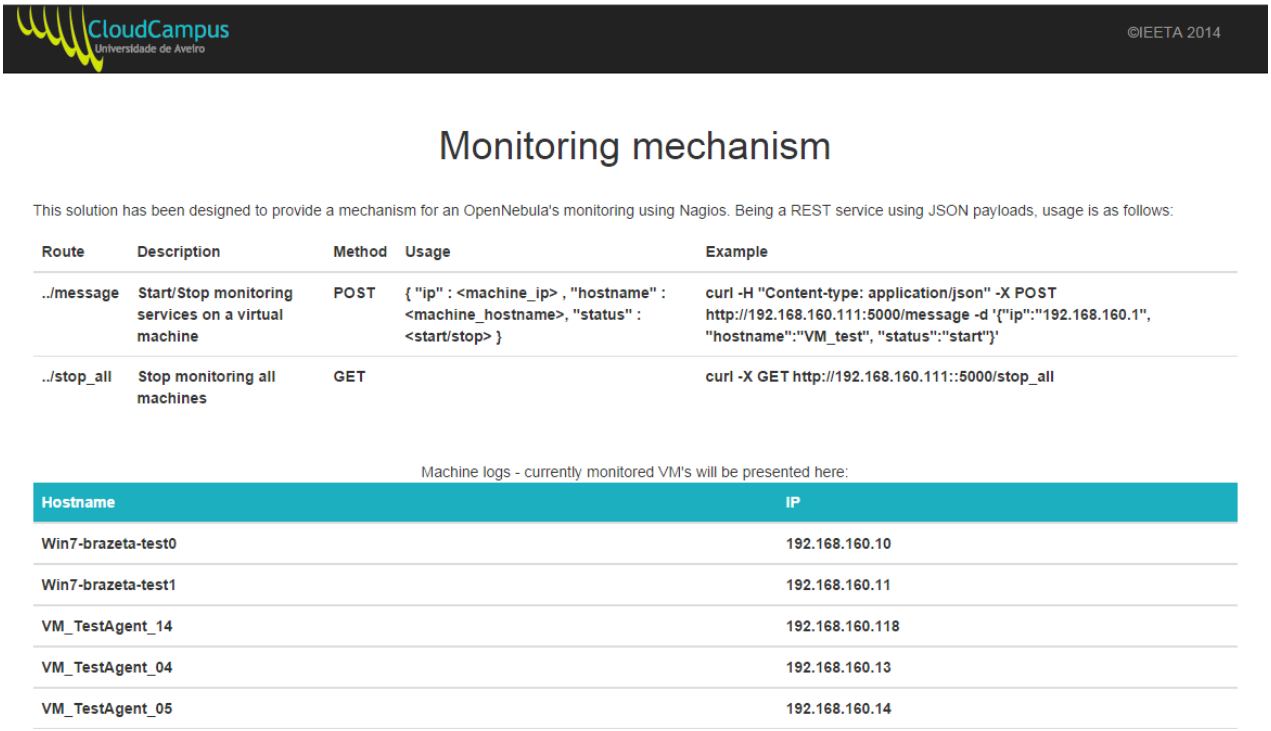

Figure 6.27 – Web interface for python program used for Nagios assistance

As Nagios did not have a mechanism to know what VMs exist and where they are hosted, a small application was also developed to keep Nagios informed. The python program, `cloud_capture.py` on Appendix A4, is fed by the startup/shutdown script of the VMs, which send an HTTP REST start request to the listening port (on our case 5000). The program inserts or updates the VM data on the database and creates a Nagios machine file, which will be read by Nagios. When the machine is shutdown, the stop request is sent and the data deleted. This procedure ensures an optimal control over the virtual infrastructure as well. Figure 6.27 is a screenshot of the program's web interface, listing monitored VMs.

This specific monitoring addon was developed due to the necessity to know the VMs status at all times. Ocasionally, and even though they are up and running for the IaaS, the VMs can be shutdown, stuck or having their network down. This can occur due to several factors, for instance when a user reboots the machine. There was no way to make this validation dynamically, so we experimented Nagios for virtual monitoring. The Nagios integration was not a plug & play incorporation and we had some particular vicissitudes, as the fast deployment/undeployment of VMs.

Chapter 7

7 Conclusions

The growing maturity of Cloud Computing has motivated new ways of research: users nowadays can control very powerful resources and can expand their investigation beyond local limits. This process also works as reverse engineering as it compels Cloud providers and its ecosystem to evolve constantly.

This dissertation, focusing on research of an agile infrastructure to support HPC applications in the Cloud, ended up in a flexible architecture and consistent implementation named CloudCampus. Starting from a deep understanding of protocols, requirements, market shares and technology mechanisms, we proposed a modeling architecture for the system, implemented it with improvements. Then we demonstrated that the system is an added value to HPC researchers and users, amongst other clients. CloudCampus is built around a model that abstracts the scattered aggregated hardware resources, and introduces simple management tools. While leveraging advanced resources in virtualization technology, it seamlessly provides a highly aware scheduler that optimizes and prioritizes requests automatically, according to configuration and available resources.

The work in this thesis was motivated by the users' needs and application challenges, leading to the pursuit for a sustainable yet powerful infrastructure. For this purpose, a set of specific objectives was proposed in order to succeed:

Bringing innovative technical solutions on the Cloud for research applications

We created new solutions for research applications on the Cloud, both on conceptual and pratical approaches. We conceived a Sky Computing architecture capable of aggregating hardware resources to join and combine different Cloud providers. We tested and chose from the available open source software the composing parts of the model and one by one put them to work together. After many experiments and updates, some improvements were made to enhance the system, such as performance tuning and monitoring.

Leveraging workload mobility within the hybrid Cloud

We leveraged workload mobility within the built hybrid Cloud, sharing data with public and private computing nodes seamlessly to obtain the needed output. Our Sky Computing architecture is able to join and combine different Cloud providers, either private or public, and seamlessly deliver resources upon the above layers' requests, allowing monitoring and accounting for possible charging purposes.

Developing a small budget solution for intensive computing

We established a small budget solution for intensive computing, using existing on-premise resources primarily and bursting to public resources on demand. We leveraged several application scenarios, including real-world applications, for a proof-concept accomplishment. In collaboration with the UA's Department of Physics, we acquired their traditional model for processing the daily weather forecast and migrated it to the Cloud. The experiment made us realize there was much to tune, either from technical issues, as disk block size, or infrastructure policies, as the campus firewall rules. In another collaboration with the UA's Bioinformatics Research Group, we also grown insight and created new methodologies for their research. The others experiments, as extending the infrastructure to small hardware with all the benchmarks, tests and applications were also a valueable asset to future work with budget constrains.

Providing an agile way to manage all the gathered resources

We provided an agile way to manage all gathered resources, delivering a centralized web application which encompasses all assets. We made the necessary improvements, federated authentication (Shibboleth), custom scheduling and monitoring (Nagios), and moved beyond HPC, opening the door to other research opportunities that arose with the platform build – CloudCampus. The constructed infrastructure is stable and steady, leaving one more useful tool to researchers and users on UA's campus to use.

Having implemented this system and extensively tested its performance and functionality on a real-life scenarios, our results validate the feasibility of CloudCampus' architecture and that it is thorough and can be effectively extended to provide more resources to applications. It was also demonstrated that CloudCampus can be a valuable asset for developing systems to improve intensive computing.

7.1 Future developments

With the work developed on this dissertation, we can point out a number of improvements and new paths for future developments. Our immediate priority is to continue actively participating in the development and enhancement of CloudCampus, increasingly adding computing capacity and performance. The system limits, mostly Cloud-driven as memory aggregation, can be analyzed to be extended or solved.

One of the most interesting areas to study is how to effectively protect Cloud data and privacy, which was not the focus on this work. While still maturing, it is also fascinating to follow the directions of the Cloud's development, currently driven by big data and environmental performance. For further reading on this topic please refer to citations [152], [165].

7.2 Final considerations

The CloudCampus system is still a small internal production, held on IEETA. Even with the adition of tens of lab computers to our two HP Proliant servers and one Dell, this is effectively a small Cloud. The problems are diverse: servers are being used traditionally and do not want to migrate; labs have classes and other batch tests running; or simply an infrastructure like this cannot be managed by a student. Also the three blade servers are not dedicated to a single use, they are continuously used during both semesters to support classes and almost every day with overcommitted capacity.

These constraints made the work a little harder, more challenging and more difficult to run tasks and tests, but we believe that this can be solved in a near future. Currently effforts are being made to expand CloudCampus to duplicate its capacity. Once the infrastructure is settled, it is possible to better fulfill the system's goals. Considering the heterogeneous workloads and different client hardware specifications, the servers should be categorized in clusters, with the best hardware saved for higher demanding computing capacity demand. At this point, the infrastructure should be more than prepared for future research trends.

7.3 Contributions

In addition to this document, this thesis' work resulted on several contributions to the scientific community, either on technical guides and reports, conference proceedings, journals, open-source software, among others which do not have visibility. The work related to this dissertation is described below on the following paragraphs.

The first article a) emerged during the state-of-the-art phase and was induced by the tottering development of the Cloud, which resulted on an extensive listing of providers, protocols, standards, software and the recommendations for a future path. Article b) reflected the first architecture draft, a new model proposed to fill the needs of the Cloud aggregation and a novelty on hybrid Clouds at the time. The work made until this point was also an advantage to the paper e), e), which focused on the PaaS layer but was complemented with a new perspective of the Cloud, with aggregation of different source providers into a hybrid infrastructure. After the model was built, paper c) was a statement of that effort, shared with the community. It described composing layers' software, tests and enhancements, practical implemented scenarios and results. On a more specific conference, we presented the paper d) with the model built before applied to a weather forecast problem, computed by a HPC method. We approached the problem insight, the Cloud migration and the outcome results. We continued the related work and published an expanded version of this paper on an international journal f), covering more variables, such as low-budget hardware computing, and making comparison with traditional solutions. Finally, the work on the Sky Computing infrastructure also lead to an invitation to extend the article on another journal, 0, on which we included the real-world applications tests and conclusions.

Articles in conference proceedings

a) André Monteiro, Joaquim S. Pinto, Cláudio Teixeira, Tiago Simões Batista. *Cloud Interchangeability - Redefining Expectations*. CLOSER 2011 - International Conference on Cloud Computing and Services Science, Noordwijkerhout, The Netherlands, May 2011, DOI 10.5220/0003393301800183

b) André Monteiro, Joaquim S. Pinto, Cláudio Teixeira, Tiago Simões Batista. *Sky Computing – Exploring the aggregated Cloud resources – part I*. CISTI2011 - 6ª Conferência Ibérica de Sistemas e Tecnologias de Informação Chaves, Chaves, Portugal, June 2011, ISBN 978-1-4577-1487-0

c) André Monteiro, Cláudio Teixeira, Joaquim S. Pinto. *Sky Computing – Exploring the aggregated Cloud resources – part II*. CISTI 2014 - 9ª Conferencia Ibérica de Sistemas y Tecnologías de Información, Barcelona, vol. I, p. 798-803, June 2014, DOI 10.1109/CISTI.2014.6876862

d) André Monteiro, Cláudio Teixeira, Joaquim S. Pinto. *Migrating HPC applications to the Cloud: a use case of weather forecast*. IBERGRID 2014 - 8th Iberian Grid Infrastructure Conference, Aveiro, vol. I, p. 115-128, September 2014, ISBN 978-84-9048-246-9

Articles in international journals

e) Cláudio Teixeira, Joaquim S. Pinto, Ricardo Azevedo, Tiago Simões Batista, André Monteiro. *The Building Blocks of a PaaS. International Journal of Network Management*, vol. 22, no. 1, p. 75-99, January 2013, DOI 10.1007/s10922-012-9260-2

f) André Monteiro, Cláudio Teixeira, Joaquim S. Pinto. *HPC in Weather Forecast - Moving to the Cloud*. International Journal of Cloud Applications and Computing, vol. 5, no. 1, p. 14-31, March 2015, DOI 10.4018/ijcac.2015010102

g) André Monteiro, Cláudio Teixeira, Joaquim S. Pinto. *Sky Computing: exploring the aggregated Cloud resources*. Journal of Networks, Software Tools and Applications - Special Issue on Cloud Computing and Internet of Things of the Cluster Computing, May 2015 (submitted as invited author)

Other contributions

Another colaborations were made, with focus on the open-source IaaS OpenNebula. We were testers and developers since the version 1.6 (2010) to current version 4.1 (2015), but contributions on code remain with GPU license. Our custom IaaS driven from OpenNebula, CloudCampus, is stable and implemented on UA, h). The Windows contextualization guide is the 3rd most visited page on IEETA since 2010 with 44,531 views, after “Current events” and “Main page” i). The OpenNebula blog j) was spin-off from the IEETA guide, suited for OpenNebula users. With the evolution of OpenNebula we developed an add-on for Windows contextualization, k). Also related to OpenNebula, we were the main Portuguese translators since version 2011, as in l).

h) CloudCampus (internal use only)
   - http://cloudcampus.ua.pt or http://cloudpt2.clients.ua.pt
i) IEETA Wiki
   - http://wiki.ieeta.pt/wiki/index.php/OpenNebula
j) OpenNebula Blog
   - http://opennebula.org/configuring-virtual-machines-with-windows-templates
k) OpenNebula Add-ons
   - https://github.com/OpenNebula/addon-context-windows
l) Transifex PT translator for OpenNebula
   - https://www.transifex.com/organization/opennebula/dashboard/all_projects/pt_PT

References

[1] H. AlHakami, “Comparison Between Cloud and Grid Computing: Review Paper,” *Int. J. Cloud Comput. Serv. Archit.*, vol. 2, no. 4, pp. 1–21, Aug. 2012.

[2] Google Inc., “Google Trends.” [Online]. Available: http://www.google.com/trends.

[3] A. Mohammed, “Explore resources within the,” *Comput. Wkly.*, pp. 18–20, 2009.

[4] J. Hoopes, *Virtualization for Security: Including Sandboxing, Disaster Recovery, High Availability, Forensic Analysis, and Honeypotting*. Syngress, 2008.

[5] G. Pallis, “Cloud Computing: The New Frontier of Internet Computing,” *Internet Comput. IEEE*, vol. 14, no. 5, pp. 70–73, 2010.

[6] M. Pokharel and J. S. Park, “Cloud computing,” in *Proceedings of the 3rd International Conference on Theory and Practice of Electronic Governance - ICEGOV ’09*, 2009, p. 409.

[7] C. Evangelinos and C. Hill, “Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon’s EC2,” *October*, vol. 2, no. 2.40, pp. 2–34, 2008.

[8] C. Teixeira, R. Azevedo, J. S. Pinto, and T. Batista, “User Provided Cloud Computing,” in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, 2010, pp. 727–732.

[9] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, “Case study for running HPC applications in public clouds,” *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, Chicago, Illinois, pp. 395–401, 2010.

[10] A. Marathe, R. Harris, D. K. Lowenthal, B. R. de Supinski, B. Rountree, M. Schulz, and X. Yuan, “A Comparative Study of High-Performance Computing on the Cloud,” in *22nd international symposium on High-performance parallel and distributed computing (HPDC ’13)*, 2013, pp. 239–250.

[11] 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,” *Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM, Bolton Landing, NY, USA, pp. 164–177, 2003.

[12] J. E. Simons and J. Buell, “Virtualizing high performance computing,” *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 4, pp. 136–145, 2010.

[13] M. D. de Assuncao, A. di Costanzo, and R. Buyya, “Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters,” in *Proceedings of the 18th ACM international symposium on High performance distributed computing - HPDC ’09*, 2009, p. 141.

[14] C. Jianhua, H. Qinming, G. Qinghua, and H. Dawei, “Performance Measuring and Comparing of Virtual Machine Monitors,” in *Embedded and Ubiquitous Computing, 2008. EUC ’08. IEEE/IFIP International Conference on*, 2008, vol. 2, pp. 381–386.

[15] NIST, *Guide to Security for Full Virtualization Technologies*, no. SP 800–125. DIANE Publishing, 2011.

[16] “LCX - Linux Containers.” [Online]. Available: https://linuxcontainers.org/.

[17] B. Russell, “Passive Benchmarking with docker LXC, KVM & OpenStack,” 2014.

[18] A. Williams, “Spoof Video Symbolizes The Energy And Brashness Of OpenStack, A Rising Cloud Power,” 2013. [Online]. Available: http://techcrunch.com/2013/04/21/dope-n-stack-e-n-t-e-r-p-r-i-s-e-video-shows-the-openstack-cloud/.

[19] I. Red Hat, “Kernel Based Virtual Machine,” 2013. [Online]. Available: http://www.linux-kvm.org.
[20] “Xen.” [Online]. Available: http://wiki.xen.org.
[21] Docker, “Docker - Build, Ship and Run Any App Anywhere.” [Online]. Available: http://www.docker.com/.
[22] C. Chaubal, “The architecture of VMware eSXi.” VMware, 2007.
[23] T. G. Peter Mell, “The NIST Definition of Cloud Computing,” vol. 145, no. 6, p. 3, 2011.
[24] C. Bunch and C. Krintz, “Enabling automated HPC / database deployment via the appscale hybrid cloud platform,” *Proceedings of the first annual workshop on High performance computing meets databases*. ACM, Seattle, Washington, USA, pp. 13–16, 2011.
[25] Amazon Web Services LLC, “AWS | Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting.” [Online]. Available: http://aws.amazon.com/ec2.
[26] I. CloudBees, “CloudBees: The Java PaaS Company.” [Online]. Available: http://www.cloudbees.com/.
[27] G. Inc., “Google App Engine - Google Developers.” [Online]. Available: https://developers.google.com/appengine/.
[28] IBM, “IBM Cloud Computing: Platform as a Service (PaaS).” [Online]. Available: http://www.ibm.com/cloud-computing/us/en/paas.html.
[29] Jelastic, “Platform-as-Infrastructure Provider - Jelastic.” [Online]. Available: http://jelastic.com/.
[30] M. Corporation, “Windows Azure: Microsoft Cloud Platform.” [Online]. Available: http://www.windowsazure.com/.
[31] O. Inc, “OutSystems Platform - High Productivity PaaS.” [Online]. Available: http://www.outsystems.com/platform/.
[32] I. Red Hat, “OpenShift by Red Hat.” [Online]. Available: https://www.openshift.com/.
[33] S. co. Inc., “What is Platform-as-a-Service - Salesforce.com.” [Online]. Available: http://www.salesforce.com/paas/overview/.
[34] H. Inc., “Heroku | Cloud Application Platform.” [Online]. Available: https://www.heroku.com/.
[35] P. S. Inc., “Welcome to CloudFoundry.” [Online]. Available: http://www.cloudfoundry.com/.
[36] WSO2, “WSO2 Stratos - 100% Open Source Cloud Platform.” [Online]. Available: http://wso2.com/cloud/stratos/.
[37] ActiveState Software Inc, “Stackato | The secure cloud application platform for the enterprise | ActiveState.” [Online]. Available: http://www.activestate.com/stackato.
[38] CumuLogic Inc., “Build AWS-compatible Private Clouds with CumuLogic Platform-as-a-Service.” [Online]. Available: http://www.cumulogic.com/cumulogic-software-portfolio/paas/.
[39] Gigaspaces, “Cloudify - Manage your Complex, DevOps Applications on OpenStack Public, Private and Hybrid Cloud | GigaSpaces.” [Online]. Available: http://www.gigaspaces.com/cloudify-devops-cloud-application-management/meet-cloudify.
[40] R. R. Expósito, G. L. Taboada, S. Ramos, J. Touriño, and R. Doallo, “Performance analysis of HPC applications in the cloud,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 1, pp. 218–229, 2013.
[41] “Joyent vs. Amazon Web Services.” [Online]. Available: http://joyent.com/products/compute-service/aws-comparison.
[42] Eucalyptus Systems Inc., “The Eucalyptus Cloud.” [Online]. Available: http://www.eucalyptus.com/eucalyptus-cloud/iaas.
[43] C12G, “The Open Source Toolkit for Data Center Virtualization,” 2013. [Online]. Available: http://opennebula.org/.
[44] OpenQRM, “OpenQRM | Leading Open Source Data Center and Computing Platform.” [Online]. Available: http://www.openqrm-enterprise.com/.
[45] O. Corporation, “Oracle® VM Server User’s Guide Release 2.1,” 2013. [Online]. Available: http://docs.oracle.com/cd/E11081_01/doc/doc.21/e10898/intro.htm.
[46] O. Corporation, “Oracle VM 3: Architecture and Technical Overview.” 2013.
[47] The Apache Software Foundation, “Apache CloudStack - Open Source Cloud Computing Software.” [Online]. Available: http://cloudstack.apache.org/.
[48] A. Chierici and R. Veraldi, “A quantitative comparison between xen and kvm,” *J. Phys. Conf. Ser.*, vol. 219, no. 4, p. 042005, Apr. 2010.

[49] S. Spector, “Latest Comparison of Leading Open Source Cloud Communities.” [Online]. Available: http://www.hpcloud.com/blog/latest-comparison-leading-open-source-cloud-communities.
[50] T. H. Lydia Leong, Douglas Toombs, Bob Gill, Gregor Petri, “Magic Quadrant for Cloud Infrastructure as a Service,” *Gartner*, 2014. [Online]. Available: http://www.gartner.com/technology/reprints.do?id=1-1UKQQA6&ct=140528&st=sb.
[51] R. Gelber, “US Cloud Providers Struggle With Data Privacy Laws,” *HPC in The Cloud*, 2012. [Online]. Available: http://www.hpcinthecloud.com/hpccloud/2012-05-08/us_cloud_providers_struggle_with_data_privacy_laws.html.
[52] E. Comission, “Data Protection Working Party, Opinion 05/2012 on Cloud Computing.”
[53] F. Araujo, S. Boychenko, R. Barbosa, and A. Casimiro, “Replica placement to mitigate attacks on clouds,” *J. Internet Serv. Appl.*, vol. 5, 2014.
[54] B. Hudzia, “Memory Aggregation For KVM Hecatonchire Project,” 2012. [Online]. Available: http://www.linux-kvm.org/wiki/images/2/29/2012-forum-Hecatonchire_Benoit_Hudzia.pdf.
[55] A. Anbar, V. K. Narayana, and T. El-Ghazawi, “Distributed Shared Memory Programming in the Cloud,” in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, 2012, pp. 707–708.
[56] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazi, #232, res, S. Mitra, A. Narayanan, D. Ongaro, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, “The case for RAMCloud,” *Commun. ACM*, vol. 54, no. 7, pp. 121–130, 2011.
[57] T. Inc., “Terracotta BigMemory In-Memory Data Management for the Enterprise.” [Online]. Available: http://terracotta.org/products/bigmemory.
[58] I. VMware, “GemFire Enterprise 6.5.” [Online]. Available: http://community.gemstone.com/display/gemfire.
[59] T. A. S. Foundation, “Hadoop.” [Online]. Available: http://hadoop.apache.org/.
[60] M. P. I. Forum, “Message Passing Interface.” [Online]. Available: http://www.mpi-forum.org/.
[61] A. B. and A. Shiloh., “M O S I X - Cluster Operating System .” [Online]. Available: http://www.mosix.cs.huji.ac.il/.
[62] “The OpenMP® API specification for parallel programming.” [Online]. Available: http://openmp.org.
[63] M. Corporation, “Dryad - Microsoft Research.” [Online]. Available: http://research.microsoft.com/research/sv/dryad/.
[64] G. Cook, “How Clean is Your Cloud?,” Greenpeace International, 2012.
[65] S. Vobugari, D. V. L. N. Somayajulu, B. M. Subraya, and M. K. Srinivasan, “A Roadmap on Improved Performance-centric Cloud Storage Estimation Approach for Database System Deployment in Cloud Environment,” in *Proceedings of the 2013 IEEE 14th International Conference on Mobile Data Management - Volume 02*, 2013, pp. 182–187.
[66] G. Caryer, T. Rings, J. Gallop, S. Schulz, J. Grabowski, I. Stokes-Rees, and T. Kovacikova, “Grid/cloud computing interoperability, standardization and the NGN,” in *Intelligence in Next Generation Networks ICIN 2009 13th International Conference on* , 2009, p. 6.
[67] R. L. Grossman, “The Case for Cloud Computing,” *IT Professional*, vol. 11, no. 2, p. 4, 2009.
[68] R. Grossman, M. R. Berthold, E. Marcadé, R. Pechter, M. Hoskins, W. Thompson, and R. Holada, “Open Standards and Cloud Computing KDD-2009 Panel Report,” *Exch. Organ. Behav. Teach. J.*, pp. 11–18, 2009.
[69] C. A. Lee, “A perspective on scientific cloud computing,” *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, Chicago, Illinois, pp. 451–459, 2010.
[70] C. Hoi and C. Trieu, “Ranking and mapping of applications to cloud computing services by SVD,” in *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, 2010, pp. 362–369.
[71] “Cloud Computing Standards,” *J. Inf. Technol. Manag.*, vol. 25, no. 8, p. 44, 2012.
[72] T. Rings, J. Grabowski, and S. Schulz, “On the Standardization of a Testing Framework for Application Deployment on Grid and Cloud Infrastructures,” in *Advances in System Testing and Validation Lifecycle (VALID), 2010 Second International Conference on*, 2010, pp. 99–107.
[73] I. C. Society, “IEEE Cloud Computing Standards Study Group,” 2010. [Online]. Available: http://www.computer.org/portal/web/sab/cloud.

[74] I. C. Society, “Standards in Cloud Computing IEEE Standards Association.” [Online]. Available: http://cloudcomputing.ieee.org/standards.
[75] DMTF, “DMTF to Develop Standards for Managing a Cloud Computing Environment,” 2009. [Online]. Available: http://dmtf.org/news/pr/2009/4/dmtf-develop-standards-managing-cloud-computing-environment.
[76] DMTF, “Open Virtualization Format (OVF).” [Online]. Available: http://dmtf.org/standards/ovf.
[77] S. L. GmbH, “SUSE Linux Enterprise Server 11 SP1 for x86 open-ovf.” [Online]. Available: https://www.suse.com/LinuxPackages/packageRouter.jsp?product=server&version=11&service_pack=sp1&architecture=i386&package_name=open-ovf.
[78] cloud-standards.org, “CloudStandards.” [Online]. Available: http://cloud-standards.org/.
[79] J. Dongarra, “Trends in high performance computing: a historical overview and examination of future developments,” *IEEE Circuits and Devices Magazine*, vol. 22, no. 1. pp. 22–27, 2006.
[80] J. Dongarra, “An overview of HPC and challenges for the future,” in *HPC Asia*, 2009.
[81] L. Wang, J. Zhan, W. Shi, Y. Liang, and L. Yuan, “In cloud, do MTC or HTC service providers benefit from the economies of scale?,” in *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers - MTAGS ’09*, 2009, pp. 1–10.
[82] A. Friedley and A. Lumsdaine, “Communication Optimization Beyond MPI,” *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. IEEE, pp. 2018–2021, 2011.
[83] Blaise Barney, “Message Passing Interface (MPI).” [Online]. Available: https://computing.llnl.gov/tutorials/mpi/.
[84] Blaise Barney, “OpenMP.” [Online]. Available: https://computing.llnl.gov/tutorials/openMP/.
[85] R. Rolf, H. Georg, and J. Gabriele, “Hybrid MPI and OpenMP Parallel Programming,” 2013.
[86] J. Dean and S. Ghemawat, “MapReduce : Simplified Data Processing on Large Clusters,” *Commun. ACM*, vol. 51, no. 1, pp. 1–13, 2008.
[87] C. Simmendinger, “Global Address Space Programming Interface,” 2012. [Online]. Available: http://www.gaspi.de/en/project.html.
[88] D. Grunewald and C. Simmendinger, “The GASPI API specification and its implementation GPI 2.0,” in *7th International Conference on PGAS Programming Models*, 2013.
[89] S. G. I. Corp., “OpenSHMEM.” [Online]. Available: http://bongo.cs.uh.edu/.
[90] T. H. Lydia Leong, Douglas Toombs, Bob Gill, Gregor Petri, “Magic Quadrant for Cloud Infrastructure as a Service,” 2013.
[91] J. Barr, “Amazon Web Services Blog: New Amazon EC2 Instance Type - The Cluster Compute Instance,” 2010. [Online]. Available: http://aws.typepad.com/aws/2010/07/the-new-amazon-ec2-instance-type-the-cluster-compute-instance.html.
[92] P. Zaspel and M. Griebel, “Massively Parallel Fluid Simulations on Amazon’s HPC Cloud,” *2011 First Int. Symp. Netw. Cloud Comput. Appl.*, pp. 73–78, Nov. 2011.
[93] A. Burkimsher, I. Bate, and L. S. Indrusiak, “Scheduling HPC workflows for responsiveness and fairness with networking delays and inaccurate estimates of execution times,” in *Euro-Par 2013*, 2013.
[94] M. Corporation, “Microsoft Releases Windows Compute Cluster Server 2003,” 2006. [Online]. Available: http://www.microsoft.com/en-us/news/press/2006/jun06/06-09computeclusterserver2003pr.aspx.
[95] M. Corporation, “Linux Support on Windows Azure Virtual Machines.” [Online]. Available: http://support.microsoft.com/kb/2805216.
[96] R. Prodan, M. Sperk, and S. Ostermann, “Evaluating High-Performance Computing on Google App Engine,” *IEEE Softw.*, vol. 29, no. 2, pp. 52–58, 2012.
[97] Matthew Roberts, “IBM CloudBurst Product Overview.” [Online]. Available: https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/IBM CloudBurst/page/IBM CloudBurst Product Overview.
[98] P. Computing, “HPC Cloud Service - Penguin’s On-Demand HPC Cloud Service,” 2013. [Online]. Available: http://www.penguincomputing.com/services/hpc-cloud/pod.
[99] CloudSigma, “CloudSigma - Cloud servers on a powerful IaaS platform.” [Online]. Available: http://www.cloudsigma.com.
[100] I. RackSpace Cloud, “Rackspace: We manage your cloud. You run your business.” [Online]. Available: http://www.rackspace.com/. [Accessed: 01-Nov-2014].

[101] J. Barr, “Next Generation Cluster Computing on Amazon EC2 - The CC2 Instance Type,” 2011. [Online]. Available: http://aws.typepad.com/aws/2011/11/next-generation-cluster-computing-on-amazon-ec2-the-cc2-instance-type.html.
[102] A. G. Carlyle, S. L. Harrell, and P. M. Smith, “Cost-Effective HPC: The Community or the Cloud?,” in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 169–176.
[103] R. van der Meulen, “Gartner Outlines Five Cloud Computing Trends That Will Affect Cloud Strategy Through 2015,” 2012. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1971515.
[104] J. Simons, “High Performance Computing with vSphere 5.1,” 2012. [Online]. Available: http://cto.vmware.com/high-performance-computing-with-vsphere-5-1/.
[105] J. S. and P. Z. Qasim Ali, Vladimir Kiriansky, “Performance Evaluation of HPC Benchmarks on VMware’s ESXi Server.”
[106] I. Nimbix, “Nimbix Accelerated Compute Cloud.” [Online]. Available: http://www.nimbix.net.
[107] Univa, “Univa Products: UniCloud Technical Specifications.” [Online]. Available: http://www.univa.com/products/unicloud-tech-specs.php.
[108] K. Katarzyna, “Sky Computing,” *Internet Computing, IEEE*, vol. 13, pp. 43–51, 2009.
[109] I. M. Llorente, “Eucalyptus, CloudStack, OpenStack and OpenNebula: a Tale of Two Cloud Models.” [Online]. Available: http://blog.opennebula.org/?p=4042.
[110] The Apache Software Foundation, “libcloud - a unified interface to the cloud,” 2011. [Online]. Available: http://incubator.apache.org/libcloud.
[111] Red Hat Inc., “Deltacloud,” 2010. [Online]. Available: http://deltacloud.org.
[112] T. A. S. Foundation, “jclouds - Multicloud Library,” 2014. [Online]. Available: http://code.google.com/p/jclouds.
[113] W. Beary, “fog,” 2011. [Online]. Available: http://github.com/geemus/fog.
[114] Abiquo Inc., “abiquos - The Enterprise Cloud Management Platform.” [Online]. Available: http://www.abiquo.com/.
[115] Kaavo, “Kaavo - Cloud Management Software.” [Online]. Available: http://www.kaavo.com/.
[116] Dell Inc., “Enstratius - The Enterprise Cloud Management Solution.” [Online]. Available: http://www.enstratius.com/.
[117] M. I. Cloud Kick Cloud, “Cloud Kick Cloud,” 2011. [Online]. Available: www.cloudkick.com.
[118] Red Hat, “The Aeolus Project.” [Online]. Available: http://www.aeolusproject.org/.
[119] T. McGee, “Red Hat Deltacloud Aspires to Become an Industry Standard,” 2010. [Online]. Available: http://www.linux-magazine.com/Online/News/Red-Hat-Deltacloud-Aspires-to-Become-an-Industry-Standard?category=13398.
[120] Nagios Enterprises, “Nagios - The Industry Standard In IT Infrastructure Monitoring.” [Online]. Available: http://www.nagios.org/.
[121] University of California, “Ganglia Monitoring System.” [Online]. Available: http://ganglia.sourceforge.net/.
[122] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, *Virtual Infrastructure Management in Private and Hybrid Clouds*, vol. 13, no. 5. IEEE Educational Activities Department, 2009, pp. 14–22.
[123] University of Victoria High Energy Physics Research Computing group, “Cloud Scheduler.” [Online]. Available: http://cloudscheduler.org/.
[124] R. Montero, “Managing CERN: OpenNebula Team Tames the Cloud,” 2011. [Online]. Available: http://www.hpcinthecloud.com/features/Managing-CERN-OpenNebula-Team-Tames-the-Cloud-114360459.html.
[125] I. Lorente, “Deltacloud and Libcloud drivers for OpenNebula,” 2011. [Online]. Available: http://blog.opennebula.org/?p=342.
[126] G. Juve, E. Deelman, G. B. Berriman, B. P. Berman, and P. Maechling, “An Evaluation of the Cost and Performance of Scientific Workflows on Amazon EC2,” *J. Grid Comput.*, vol. 10, no. 1, pp. 5–21, 2012.
[127] S. Saini, S. Heistand, H. Jin, J. Chang, R. Hood, P. Mehrotra, and R. Biswas, “An Application-based Performance Evaluation of NASA’s Nebula Cloud Computing Platform,” in *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, 2012, pp. 336–343.

[128] P. Jakovits and S. N. Srirama, “Adapting Scientific Applications to Cloud by Using Distributed Computing Frameworks,” in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, 2013, pp. 164–167.
[129] W. C. Skamarock Klemp, J. 420 B., Dudhia, J., Gill, J., Barke, D. M., Wang, W. and Powers, J. G., “A description of the Advance Reaserch WRF version,” 2005.
[130] “1000 Genomes - A Deep Catalog of Human Genetic Variation.” [Online]. Available: http://www.1000genomes.org/.
[131] F. W. Zwiers and H. von Storch, “On the role of statistics in climate research,” *Int. J. Climatol.*, vol. 24, no. 6, pp. 665–680, May 2004.
[132] J. Michalakes, R. Loft, and A. Bourgeois, “Performance-Portability and the Weather Research and Forecast Model,” in *HPC Asia*, 2001.
[133] D. Carvalho, A. Rocha, M. Gómez-Gesteira, and C. Santos, “A sensitivity study of the {WRF} model in wind simulation for an area of high wind energy,” *Environ. Model. Softw.*, vol. 33, no. 0, pp. 23–34, 2012.
[134] J. Dudhia, D. O. Gill, T. B. Henderson, J. B. Klemp, W. Skamarock, and W. Wang, “The Weather Research and Forecast Model: software architecture and performance,” in *11th Workshop on the Use of High Performance Computing in Meteorology*, 2011, p. 6.
[135] GMCUA, “CliM@UA,” 2013. [Online]. Available: http://climetua.fis.ua.pt.
[136] N. Huber Marcel Von Quast, Michael Hauck, and Samuel Kounev, “Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments,” *CLOSER 2011*. 2011.
[137] Fabrice Bellard, “Open source processor emulator.” [Online]. Available: http://wiki.qemu.org.
[138] I. Red Hat, “Frontpage | Gluster Community Website.” [Online]. Available: http://www.gluster.org/.
[139] G. SA, “Features, Architecture and Requirements :: MooseFS.” [Online]. Available: http://www.moosefs.org/.
[140] William D. Norcott, “IOzone Filesystem Benchmark,” 2013. [Online]. Available: http://www.iozone.org.
[141] L. P. Hewlett-Packard Development Company, “HP ProLiant BL460c Gen8 E5-2620v2 2P 32GB-R P220i/512 FBWC Server/S-Buy.” [Online]. Available: http://shopping1.hp.com/is-bin/INTERSHOP.enfinity/WFS/WW-USSMBPublicStore-Site/en_US/-/USD/ViewProductDetail-Start;pgid=jDJwlVlq2W9SR0Yk2kO1Yuen00005nBoy9ML;sid=pTRu5KilayFt5Pxq7kdwbXCqdzpWleEO0So=?ProductUUID=MzsQ7EN5OXMAAAE_x8xevcsl&CatalogCategoryID=.R. [Accessed: 01-Jan-2015].
[142] L. P. Hewlett-Packard Development Company, “HP ProLiant BL460c Gen8 E5-2650v2 1P 32GB-R P220i/512 FBWC Server.” [Online]. Available: http://www8.hp.com/pt/pt/products/proliant-servers/product-detail.html?oid=5409979#!tab=specs. [Accessed: 01-Nov-2014].
[143] J. Beckett and R. Bradfield, “Power Efficiency Comparison of Enterprise-Class Blade Servers and Enclosures,” 2011.
[144] R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, “An elasticity model for High Throughput Computing clusters,” *J. Parallel Distrib. Comput.*, vol. 71, no. 6, pp. 750–757, 2011.
[145] L. Gillam, B. Li, J. O¿Loughlin, and A. P. Tomar, “Fair Benchmarking for Cloud Computing systems,” *J. Cloud Comput. Adv. Syst. Appl.*, vol. 2, no. 1, p. 6, 2013.
[146] V. Glenis, A. S. McGough, V. Kutija, C. Kilsby, and S. Woodman, “Flood modelling for cities using Cloud computing,” *J. Cloud Comput. Adv. Syst. Appl.*, vol. 2, no. 1, p. 7, 2013.
[147] P. Nowoczynski, J. Sommerfield, J. Yanovich, J. R. Scott, Z. Zhang, and M. Levine, “The data supercell,” *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the campus and beyond*. ACM, Chicago, Illinois, pp. 1–11, 2012.
[148] J. L. Lucas Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, “Dynamic placement of virtual machines for cost optimization in multi-cloud environments,” in *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, 2011, pp. 1–7.

[149] André Monteiro, Joaquim Sousa Pinto, Cláudio Teixeira, and Tiago Batista, “Sky Computing: Exploring the aggregated Cloud resources - part I,” in *CISTI2011: 6ª Conferência Ibérica de Sistemas e Tecnologias de Informação*, 2011, vol. II, pp. 183–186.
[150] André Monteiro, “Configuring Virtual Machines with Windows Templates,” 2011. [Online]. Available: http://opennebula.org/configuring-virtual-machines-with-windows-templates/.
[151] A. Monteiro, “Managing Cloud Computing with OpenNebula,” 2013. [Online]. Available: http://wiki.ieeta.pt/wiki/index.php/OpenNebula.
[152] S. K. Barker and P. Shenoy, “Empirical evaluation of latency-sensitive application performance in the cloud,” in *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, 2010, pp. 35–46.
[153] Phoronix Media, “Phoronix Test Suite - Linux Testing & Benchmarking Platform, Automated Testing Framework, Open-Source Benchmarking.” [Online]. Available: http://www.phoronix-test-suite.com/.
[154] D. Sheetz, L. Song, and A. Rikun, “Predicting the Relative Performance of CPU.,” in *Int. CMG Conference*, 2008, pp. 301–308.
[155] J. Choudhury, “Corrected model for predicting the relative performance of CPU,” *Proc. 19th High Perform. Comput. …*, vol. 31, no. 5, pp. 532–533, May 2011.
[156] M. Gusev, S. Ristov, and G. Velkoski, “Web Service Performance Analysis in Virtual Environment,” in *ICT INNOVATIONS 2013: ICT INNOVATIONS AND EDUCATION*, 2014, vol. 231, pp. 167–176.
[157] Intel Corporation, “ARK | Intel® Xeon® Processor X5675 (12M Cache, 3.06 GHz, 6.40 GT/s Intel® QPI).” [Online]. Available: http://ark.intel.com/products/52577/.
[158] Intel Corporation, “ARK | Intel® Core™ i5-2400 Processor (6M Cache, up to 3.40 GHz).” [Online]. Available: http://ark.intel.com/products/52207.
[159] SPEC, “SPECvirt_sc2010 Result: Hewlett Packard Company.” [Online]. Available: http://www.spec.org/virt_sc2010/results/res2012q4/virt_sc2010-20121002-00047-perf.html.
[160] OASIS, “UDDI | Online community for the Universal Description, Discovery, and Integration.” [Online]. Available: http://uddi.xml.org/.
[161] “Apache jUDDI.” [Online]. Available: http://juddi.apache.org.
[162] M. S., “SOA Client :: Add-ons for Firefox.” [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/soa-client/.
[163] C. Costa, “Autenticação nos sistemas informáticos da Universidade de Aveiro,” Universidade de Aveiro, 2009.
[164] Shibboleth Consortium, “Shibboleth.” [Online]. Available: http://shibboleth.net/.
[165] N. Gonzalez, C. Miers, F. Redígolo, M. Simplício, T. Carvalho, M. Näslund, and M. Pourzandi, “A quantitative analysis of current security concerns and solutions for cloud computing,” *J. Cloud Comput. Adv. Syst. Appl.*, vol. 1, no. 1, p. 11, Jul. 2012.

Appendix

A. Scripts

1. Windows contextualization files

one-context.ps1

```
#################################################################
##### Windows Powershell Script to configure OpenNebula VMs #####
#####   Created by andremonteiro@ua.pt and tsbatista@ua.pt  #####
#####        DETI/IEETA Universidade de Aveiro 2011-2014    #####
#################################################################

Set-ExecutionPolicy unrestricted -force # not needed if already done once on
the VM
[string]$computerName = "$env:computername"
[string]$ConnectionString = "WinNT://$computerName"

# Get all drives and select only the one that has "CONTEXT" as a label
$contextDrive = Get-WMIObject Win32_Volume | ? { $_.Label -eq "CONTEXT" }

# Return if no CONTEXT drive found
if ($contextDrive -eq $null) {
     Write-Host "No Context CDROM found."
               exit 1
}

function getContext($file) {
        $context = @{}
        switch -regex -file $file {
                "(.+)='(.+)'" {
                        $name,$value = $matches[1..2]
                        $context[$name] = $value
                }
        }
        return $context
}

function addLocalUser($context) {

    # Create new user
        $username =  $context["username"]
        $ADSI = [adsi]$ConnectionString

        if(!([ADSI]::Exists("WinNT://$computerName/$username"))) {
           $user = $ADSI.Create("user",$username)
           $user.setPassword($context["password"])
```

```
            $user.SetInfo()
        }
        # Already exists, change password
        else{
            $admin = [ADSI]"WinNT://$env:computername/$username"
            $admin.psbase.invoke("SetPassword", $context["PASSWORD"])
        }

    # Set Password to Never Expires
    $admin = [ADSI]"WinNT://$env:computername/$username"
    $admin.UserFlags.value = $admin.UserFlags.value -bor 0x10000
    $admin.CommitChanges()
 # Add user to local Administrators
    $groups = "Administrators", "Administradores"
    $groups = (Get-WmiObject -Class " Win32_Group" | where { $_.SID -like "S-
1-5-32-544" } | select -ExpandProperty Name)

    foreach ($grp in $groups) {
    if([ADSI]::Exists("WinNT://$computerName/$grp,group")) {
                $group = [ADSI] "WinNT://$computerName/$grp,group"
                        if([ADSI]::Exists("WinNT://$computerName/$username"))
{
                                $group.Add("WinNT://$computerName/$username")
                        }
                }
        }
}

function getIp($mac) {
    $mac = $mac.Replace("-",":")
    $octet = $mac.Split(":")
    [String] $ip = ""
    $ip += [convert]::toint32($octet[2],16)
    $ip += "."+[convert]::toint32($octet[3],16)
    $ip += "."+[convert]::toint32($octet[4],16)
    $ip += "."+[convert]::toint32($octet[5],16)
    return $ip
}

function getGateway($mac) {
    $octet = $mac.Split(":")
    [String] $ip = ""
    $ip += [convert]::toint32($octet[2],16)
    $ip += "."+[convert]::toint32($octet[3],16)
    $ip += "."+[convert]::toint32($octet[4],16)
    $ip += ".254"
    return $ip
}

function configureNetwork($context) {
    $Nics = Get-WMIObject Win32_NetworkAdapterConfiguration | where
{$_.IPEnabled -eq "TRUE" -and ($_.MACAddress)}
    foreach ($nic in $Nics) {
        [String]$mac = $nic.MACAddress
        [String]$ip = getIp($mac)
        [String]$gw = $context["GATEWAY"]

        # if gateway string is empty, calculate
        if ($context["GATEWAY"])
        {
                [String]$gw = $context["GATEWAY"]
        }
        $nic.ReleaseDHCPLease()
```

```
        $nic.EnableStatic($ip , "255.255.255.0")
        $nic.SetGateways($gw)
        $DNSServers = "193.136.172.20", "193.136.171.21"

        # if DNS string is empty, set default
        if ($context["DNS"])
        {
                $DNSServers = $context["DNS"]
        }

        $nic.SetDNSServerSearchOrder($DNSServers)
        $nic.SetDynamicDNSRegistration("TRUE")
        $nic.SetWINSServer($DNSServers[0], $DNSServers[1])
    }
}

function renameComputer($context) {
    $ComputerInfo = Get-WmiObject -Class Win32_ComputerSystem
    $ComputerInfo.rename($context["HOSTNAME"])
}

function enableRemoteDesktop()
{
  # Windows 7 only - add firewall exception for RDP
    netsh advfirewall Firewall set rule group="Remote Desktop" new enable=yes
  # Enable RDP
    $Terminal = (Get-WmiObject -Class "Win32_TerminalServiceSetting" -
Namespace root\cimv2\terminalservices).SetAllowTsConnections(1)
    return $Terminal
}

function enablePing()
{
    #Create firewall manager object
    $FWM=new-object -com hnetcfg.fwmgr

    # Get current profile
    $pro=$fwm.LocalPolicy.CurrentProfile
    $pro.IcmpSettings.AllowInboundEchoRequest=$true
}

function addReadme($context) {
  $username =  $context["USERNAME"]

  if (-not (Test-Path "C:\Users\$username\Desktop"))
  {
    New-Item "C:\Users\$username" -type directory
    New-Item "C:\Users\$username\Desktop" -type directory
  }
  $readme = $contextLetter + "README.txt"
  Copy-Item $readme "C:\Users\$username\Desktop\README.txt"}

# If folder context doesn't exist create it
if (-not (Test-Path "c:\context\")) {
    New-Item "C:\context\" -type directory
    }

# Execute script
if( -not(Test-Path "c:\context\contextualized") -and (Test-Path
$contextScriptPath)) {
    $context = @{}
    $context = getContext($contextScriptPath)
    addLocalUser($context)
```

```
    renameComputer($context)
    enableRemoteDesktop
    enablePing
 Start-Sleep -s 30
    configureNetwork($context)
    echo "contextualized" |Out-File ("c:\context\contextualized")
    restart-computer -force
}
 ## Restart a second time
elseif( -not(Test-Path "c:\context\contextualizedNetwork") -and (Test-Path
$contextScriptPath"))
{
    $context = @{}
    $context = getContext($contextScriptPath)
    configureNetwork($context)
    addReadme($context)
    echo "contextualizedNetwork" |Out-File
("c:\context\contextualizedNetwork")
}
else
{
    echo "finished configuration"
}
#####################################################################
# remember to disable password complexity requirements in computer #
# management while preparing the Cloud image!                       #
# remember to run "Set-ExecutionPolicy bypass -force" before this! #
#####################################################################
```

startup.vbs

```
strComputer = "."

Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")

Set colItems = objWMIService.ExecQuery _
    ("Select * From Win32_LogicalDisk Where VolumeName = 'CONTEXT'")

Dim driveLetter

For Each objItem in colItems
    driveLetter = objItem.Name
    Exit For
Next

If Len(driveLetter) Then
    contextPath = driveLetter & "\one-context.ps1"
    Set objShell = CreateObject("Wscript.Shell")
    objShell.Run("powershell -NonInteractive -NoProfile -NoLogo -
ExecutionPolicy Unrestricted -file " & contextPath)
End If


Dim objFSO, objFolder
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFSO.CreateFolder("C:\executedVBScript")
objShell.Run("slmgr /skms denethor.ua.pt:1688")
objShell.Run("slmgr /ato")
```

SetupComplete.cmd

```
del /Q /F c:\windows\system32\sysprep\unattend.xml
del /Q /F c:\windows\panther\unattend.xml
cscript //b D:\startup.vbs
```

Shutdown.cmd

```
cscript //b  D:\updateNagios.vbs "stop"
```

updateNagios.vbs

```
' Code to get Nagios server IP'

strNagios = "192.168.160.111"
fileName = "D:\context.sh"

Const ForReading = 1
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objTextFile = objFSO.OpenTextFile _
    (fileName, ForReading)

Do Until objTextFile.AtEndOfStream
    strNextLine = objTextFile.Readline
    arrServiceList = Split(strNextLine , "=")

    For i = 0 to Ubound(arrServiceList)
      If (arrServiceList(i) = "NAGIOS_IP") Then
      strNagios = Replace(arrServiceList(i+1), "'", "")
        Wscript.Echo strNagios
      ElseIf (arrServiceList(i) = "HOSTNAME") Then
      strHostname = Replace(arrServiceList(i+1), "'", "")
        Wscript.Echo strHostname

      ElseIf (arrServiceList(i) = "IP_PUBLIC") Then
      strIP = Replace(arrServiceList(i+1), "'", "")
        Wscript.Echo strIP
      End If
    Next
Loop

If WScript.Arguments.Length = 1 Then
      strFunction = WScript.Arguments.Item(0)
Else
      strFunction = "stop"
End If

strCurl = "curl -H ""Content-type: application/json"" -X POST http://" &
strNagios &":5000/message -d ""{\""ip\"":\""" & strIP &"\"",
\""hostname\"":\""" & strHostname & "\"", \""status\"":\""" & strFunction &
"\""}"""

Set objShell = CreateObject("Wscript.Shell")
objShell.Run(strCurl)
```

2. CentOS contextualization files

rc.local

```
#!/bin/sh
#
# This script will be executed *after* all the other init scripts.
# You can put your own initialization stuff in here if you do not
# want to do the full Sys V style init stuff.

# OpenNebula Contextualization

CONTEXT_DIR=/mnt/context
mkdir -p $CONTEXT_DIR
mount -t iso9660 /dev/sr0 $CONTEXT_DIR

if [ -f $CONTEXT_DIR/context.sh ]; then
  . $CONTEXT_DIR/init.sh
fi
umount $CONTEXT_DIR

touch /var/lock/subsys/local
```

init.sh

```
#!/bin/bash

if [ -f /mnt/context/context.sh ];
then
  . /mnt/context/context.sh
else
  . /media/CDROM/context.sh
fi

hostname $HOSTNAME
sed -i "/HOSTNAME=/s/=.*$/=$HOSTNAME/" /etc/sysconfig/network
EMPTY=""
DEVICENAME=$(ip -o link show | awk '{print $2,$9}' |grep ': UP' | cut -d ':' -
f 1)

if [[ -z "$DEVICENAME" ]]; then
        DEVICENAME="eth0"
fi

if [ -n "$IP_PUBLIC" ]; then
        ifconfig $DEVICENAME $IP_PUBLIC
fi

if [ -n "$NETMASK_PUBLIC" ]; then
        ifconfig eth1 netmask $NETMASK_PUBLIC
fi

if [ -n "$GATEWAY" ]; then
        ip route add default via $GATEWAY
fi

ROUTE=$(route -n | grep 'UG[ \t]' | awk '{print $2}')

if [ -n "$ROUTE" ]; then
        if [ "$ROUTE"=="$GATEWAY" ]; then
                echo "equal"
        else
                if [ -n "$GATEWAY" ]; then
```

```
                        ip route del default
                        ip route add default via $GATEWAY
                fi

        fi
else
        if [ -n "$GATEWAY" ]; then
                ip route add default via $GATEWAY
        fi
fi

if [ -f /mnt/context/resolv.conf ]; then
        cat /mnt/context/resolv.conf > /etc/resolv.conf
fi

echo "finishing configuring network" #> $file
/etc/init.d/network restart


if [ -f /mnt/$ROOT_PUBKEY ]; then
        mkdir -p /root/.ssh
        cat /mnt/$ROOT_PUBKEY >> /root/.ssh/authorized_keys
        chmod -R 600 /root/.ssh/
        chcon -R -t user_ssh_home_t /root/.ssh
fi

if [ -n "$ROOT_PW" ]; then
        passwd -d root
        echo $ROOT_PW | passwd --stdin root
fi

if [ -n "$USERNAME" ]; then
     if id -u $USERNAME > 0; then
                echo EXISTS!
     else
        useradd $USERNAME
        echo $USER_PW | passwd --stdin $USERNAME
        if [ -f /mnt/$USER_PUBKEY ]; then
                mkdir -p /home/$USERNAME/.ssh/
                cat /mnt/$USER_PUBKEY >> /home/$USERNAME/.ssh/authorized_keys
                chown -R $USERNAME:$USERNAME /home/$USERNAME/.ssh
                chmod -R 600 /home/$USERNAME/.ssh/authorized_keys
                chcon -R -t user_ssh_home_t
/home/$USERNAME/.ssh/authorized_keys
       fi
fi

chkconfig NetworkManager off
echo "finishing file" #> $file

# Install requested software
if [ -n "$SCRIPT" ]; then
        $SCRIPT
fi

curl -v -H "Content-type: application/json" -X POST
http://$NAGIOS_IP:$NAGIOS_PORT/message -d  '{"ip":"'$IP_PUBLIC'",
"hostname":"'$HOSTNAME'", "status":"start"}'

cp /mnt/onshutdown /etc/init.d/onshutdown
chmod +x  /etc/init.d/onshutdown
chkconfig --add /etc/init.d/onshutdown #--level 02356
```

onshutdown

```
#! /bin/sh
# /etc/init.d/onshutdown
# chkconfig: 2345 99 20

# Carry out specific functions when asked to by the system
case "$1" in
  start)
    echo "Starting script onshutdown "
    echo "Starting $(date)" >> /home/myshutdown.txt

    # Create lock file
    touch /var/lock/subsys/onshutdown

    # Get environment variables if not mounted
        if [ -f /mnt/context.sh ]
         then
          . /mnt/context.sh
        else
           mount -t iso9660 /dev/sr0 /mnt
          . /mnt/context.sh
        fi

    # Send post to Nagios
    curl -v -H "Content-type: application/json" -X POST http://$NAGIOS_IP:
$NAGIOS_PORT/message -d  '{"ip":"'$IP_PUBLIC'", "hostname":"'$HOSTNAME'",
"status":"start"}'

    # Umount context CD
    umount /mnt
    ;;
  stop)
    echo "Stopping script onshutdown"
    echo "Stopping $(date)" >> /home/myshutdown.txt

    # Delete the lock file
    rm -f /var/lock/subsys/onshutdown

    # Get environment variables if not mounted
    if [ -f /mnt/context.sh ]
       then
       . /mnt/context.sh
    else
       mount -t iso9660 /dev/sr0 /mnt
       . /mnt/context.sh
    fi

    # Send post to Nagios
    curl -v -H "Content-type: application/json" -X POST http://$NAGIOS_IP:
$NAGIOS_PORT/message -d  '{"ip":"'$IP_PUBLIC'", "hostname":"'$HOSTNAME'",
"status":"stop"}'

    # Umount context CD
    umount /mnt
    ;;
  *)
    echo "Usage: /etc/init.d/onshutdown {start|stop}"
    exit 2
    ;;
esac

exit 0
```

3. 1000 Genome scripts

new_script.sh

```
#!/bin/bash
#
# The algorithm is:
#    for each chromosome
#      split chromosome .vcf into 1092 individual sample .svcf files,
#                          one group at a time. (Group size is a PARAMETER!)
#      for each sample
#        patch the reference/chromosome.fa to generate the
sample/chromosome.fa
#        count words in sample/chromosome.fa
#      end
#    end
#
# The loops may be paralelized, but actions inside loop bodies are sequential!
#
# The data are in /media/comodiestore/1kgenomes/motifStats/,
# where /media/comodiestore is a NFS share:
#   192.168.166.4:/vol/F2_aggr_SATA_IEETA0_vol_Comodie/Sapiens
#
# Inside that dir there is
#   bin/                containing this script and the necessary programs
#   Reference-GRCh37/   containing the reference chromosome fasta (.fa) files
#   VCFs/               containing the .vcf files from the 1000 genomes
project
#
# The script will create 1092 results subdirectories, one per sample:
#   HG00096/
#   HG00097/
#   etc..
# Each of these will contain
#       23 .svcf files (one per chromosome)
#       23 .log files (one per chromosome)
#       23x12 .c?? files
#

START_TIME=$(date +%H:%M:%S)
echo "----------------------"
echo "START TIME" $(date +%H:%M:%S)

init=$1    # command line argument 1
final=$2   # command line argument 2
chroms="$(seq $init $final) X" #chroms="$(seq 1 22) X"
lengths="01 02 03 04 05 06 07 08 09 10 11 12"

# Expect programs to be in the same dir as this script
BIN=$(dirname $0)

# Go to base directory where data and results reside
#cd /media/comodiestore/1kgenomes/motifStats
cd /home/oneadmin/motifStats

for chrom in $chroms; do
  echo "START ZCAT" $(date +%H:%M:%S)
  vcffile=VCFs/chr$chrom.vcf.gz

  # Split multi-sample vcf files
  echo "Split multi-sample vcf files"
  let group=91    #91 # Number of samples to extract on each splitvcf
(PARAMETER)
```

```
  let min=10
  let last=$( zcat $vcffile | grep -m 1 '^#CHROM' | wc -w )
  echo "Processing $vcffile"
  samples=$( zcat $vcffile | grep -m 1 '^#CHROM' | cut -f $min- )
  install -d  $samples

  echo "START SPLIT" $(date +%H:%M:%S)
  while (( $min<=$last )); do
    let max=$min+$group-1
    echo "Splitting $vcffile with min=$min and max=$max with last=$last"

   if [ ! -e "%s/chr$chrom-%s.svcf" ]
   then
    $BIN/splitvcf $vcffile "%s/chr$chrom-%s.svcf" $min $max
    echo "Working on file %s/chr$chrom-%s.svcf"
   else
    echo "Detected existing file"
   fi
    let min+=$group
  done
  echo "END SPLIT" $(date +%H:%M:%S)
  echo "------------------------------------------------------------"

  # Patch reference chromosome with sample svcfs, and count words
  echo "Path reference chromosome with sample svcfs, and count words"
  echo "START PATCH" $(date +%H:%M:%S)
  for sample in $samples; do
    sc=$sample/chr$chrom-$sample
    echo "Processing $sc.svcf from chrom$chrom"
    $BIN/vcfpatch $sc.svcf < Reference-GRCh37/chr$chrom.fa > $sc.fa 2> $sc.log
    rm -f $sc.fa.c??
    echo "Counting fields"
    $BIN/count $sc.fa $lengths
    rm -f $sc.fa
  done
  echo "END PATCH" $(date +%H:%M:%S)
done
echo "END ZCAT" $(date +%H:%M:%S)

echo "                                                    "
echo "START TIME" $START_TIME "|         END TIME" $(date +%H:%M:%S)
```

mpirun.sh

```
#!/bin/bash

PROCESSES=22

START_TIME=$(date +%H:%M:%S)
echo "-----------------------"
echo "START TIME" $(date +%H:%M:%S)

/usr/local/bin/mpiexec.hydra -f "/mnt/data/motifStats/hydra.pt" -np $PROCESSES
/bin/bash -c "ulimit -s unlimited ; /mnt/data/motifStats/bin/new_script.sh 1
22 < /dev/null"

echo "                                                    "
echo "START TIME" $START_TIME "|         END TIME" $(date +%H:%M:%S)
```

4. Monitoring scripts

cloud_capture.py

```
import sqlite4, os
from flask import Flask, request, json, session, g, redirect, url_for, abort,
render_template, flash
from contextlib import closing

###################################
#Application name

app = Flask(__name__)
print 'Cloud capture - IEETA\'s Nagios/OpenNebula monitoring servant'

###################################
#Database connections and functions

# configuration
IP_ADDRESS='192.168.160.111'
PORT=5000
APP_PATH = '/home/ubuntu/cloud_capture/IEETA_cloud_capture/flaskr/'
DATABASE = APP_PATH + 'tmp/flaskr.db'
DEBUG = True
SECRET_KEY = 'development key'
USERNAME = 'admin'
PASSWORD = 'default'

app.config.from_object(__name__)
app.config.from_envvar('FLASKR_SETTINGS', silent=True)

#Supporting functions
def connect_db():
    """Connects to the specific database."""
    rv = sqlite3.connect(app.config['DATABASE'])
    rv.row_factory = sqlite3.Row
    return rv

def init_db():
    """Creates the database tables."""
    with app.app_context():
        db = get_db()
        with app.open_resource(app.config['APP_PATH'] + 'schema.sql',
mode='r') as f:
            db.cursor().executescript(f.read())
        db.commit()

def get_db():
    """Opens a new database connection if there is none yet for the
    current application context."""
    if not hasattr(g, 'sqlite_db'):
        g.sqlite_db = connect_db()
    return g.sqlite_db


@app.teardown_appcontext
def close_db(error):
    """Closes the database again at the end of the request."""
    if hasattr(g, 'sqlite_db'):
        g.sqlite_db.close()

###################################
#Database functions end
```

```
#../ - GET - Display index and logging

@app.route('/', methods = ['GET'])
def api_show_message():
    db = get_db()
    cur = db.execute('select hostname, ip from vm_entries order by ip asc')
    entries = cur.fetchall()
    return render_template('index.html', entries = entries)

@app.route('/message', methods = ['POST'])
def api_message():

    if request.headers['Content-Type'] == 'application/json':
        received_json = request.json

        if received_json['status'] == 'start':
            response_from_method = start_monitoring_vm(received_json['ip'],
received_json['hostname'])
        elif received_json['status'] == 'stop':
            response_from_method = stop_monitoring_vm(received_json['ip'],
received_json['hostname'])
        else:
              return "400 Bad request\n"

        return response_from_method + '\n'

    else:
        return "400 Bad request\n"

@app.route('/stop_all', methods = ['GET'])
def api_delete_all_messages():

    db = get_db()
    cur = db.execute('select hostname, ip from vm_entries order by id asc')
    entries = cur.fetchall()
    error = {}

    for row in entries:
        response_from_method = stop_monitoring_vm(row[1], row[0])

        if response_from_method.startswith('Error'):
            error[++len(error)] = response_from_method

    db = get_db()
    db.execute('delete from vm_entries')
    db.commit()

    if len(error) == 0:
        return '200 OK - All monitoring has stopped\n'
    else:
        return_string = '{"error_message":['

        for error_message in error:
            return_string = return_string + '{"' + str(error_message) + '":"'
+ error.get(error_message) + '"},'

        return_string = return_string[:-1] + ']}\n'
        return return_string

###################################
#Supporting functions

def start_monitoring_vm(vm_ip, vm_hostname):
```

```
    resulted_in = create_nagios_files(vm_ip, vm_hostname)

    if resulted_in == 'true':
        db = get_db()
        db.execute('INSERT OR REPLACE INTO vm_entries (id, hostname, ip)
VALUES ((SELECT id FROM vm_entries WHERE ip = ?),?,?);',
[vm_ip,vm_hostname,vm_ip])
        db.commit()
        return 'Cloud capture: started monitoring ' + vm_ip + " - " +
vm_hostname
    else:
        return 'Error starting ' + vm_ip + " - " + vm_hostname

def stop_monitoring_vm(vm_ip, vm_hostname):

    db = get_db()
    cur = db.execute('select hostname, ip from vm_entries where hostname =
(?) and ip = (?)', [vm_hostname,vm_ip])
    entries = cur.fetchall()

    if len(entries) != 0:
        resulted_in = delete_nagios_files(vm_hostname)

    db = get_db()
    db.execute('delete from vm_entries where hostname = (?) and ip = (?)',
[vm_hostname,vm_ip])
    db.commit()

    if resulted_in == 'true':
       return 'Cloud capture: stopped monitoring ' + vm_ip + ' - ' +
vm_hostname
    else:
       return 'Error - Monitoring stopped but cfg file not found for ' +
vm_ip + ' - ' + vm_hostname
    else:
       return 'Error - database entry not found for ' + vm_ip + ' - ' +
vm_hostname

###################################
#Nagios file handlers

def create_nagios_files(vm_ip, vm_hostname):

    try:
        f = open('nagios/' + vm_hostname + ".cfg", 'w+')
    except IOError:
        return "false"

    #File header and description
    separator = "############################################################\n"
    f.write(separator)
    f.write('# Created by IEETA\'s Cloud Capture - do not modify manually\n')
    f.write('# If need be, delete/start manually using provided
mechanisms\n')
    f.write(separator)
    f.write('\n')

    #Host definition
    f.write(separator)
    f.write('# Host definition\n')
    f.write(separator)
    f.write('\n')
    f.write('define host{\n')
```

```
    f.write('    use                     firewall-server-cloud\n')
    f.write('    host_name               ' + vm_hostname + '\n')
    f.write('    alias                   ' + vm_hostname + '\n')
    f.write('    address                 ' + vm_ip + '\n')
    f.write('    hostgroups              Cloud_Services\n')
    f.write('    contact_groups          brunobic\n')
    f.write('    }\n')
    f.write('\n')

    #Service definition
    f.write(separator)
    f.write('# Service definition\n')
    f.write(separator)
    f.write('define service{\n')
    f.write('    use                                 brunobic-service\n')
    f.write('    host_name                           ' + vm_hostname + '\n')
    f.write('    service_description                 Load\n')
    f.write('    check_command                       check_load_static\n')
    f.write('    notifications_enabled               1\n')
    f.write('    notification_interval               12000\n')
    f.write('    }\n')
    f.write('\n')
    f.write('define service{\n')
    f.write('    use                                 brunobic-service\n')
    f.write('    host_name                           ' + vm_hostname + '\n')
    f.write('    service_description                 HDD\n')
    f.write('    check_command                       check_hdd_static\n')
    f.write('    notifications_enabled               1\n')
    f.write('    notification_interval               12000\n')
    f.write('    }\n')
    f.write('\n')
    f.close()
    return "true"

def delete_nagios_files(vm_hostname):
    #delete file - how to call console operations

    if os.path.isfile('nagios/' + vm_hostname + '.cfg'):
        os.remove('nagios/' + vm_hostname + '.cfg')
        return "true"
    else:
        return "false"

#End supporting functions
if __name__ == '__main__':
    init_db()
    app.run(host=app.config['IP_ADDRESS'], port=app.config['PORT'])
```

5. CloudCampus startup scripts

installubuntu.sh

```
    #!/bin/bash

    # Install dependencies
    apt-get -qy update
    apt-get -qy install qemu-kvm libvirt0 libvirt-bin bridge-utils ruby
    install -d /usr/libexec
    ln -sf /usr/bin/qemu-system-x86_64  /usr/libexec/qemu-kvm
    echo 'KERNEL=="kvm", NAME="%k", GROUP="kvm", MODE="0660"' >
    /etc/udev/rules.d/65-kvm.rules
```

```
#Configure user
name="oneadmin"
uid="2001"
gid="$uid"
homedir="/home/$name"
shell="/bin/false"
nfsserver="fas2-
external.storage.ua.pt:/vol/F2_aggr_SATA_IEETA0_vol_CloudPT/CloudPT/2Labs"

PWLINE="$name:x:$uid:$gid:$name,,,:$homedir:$shell"

if pwline=$(getent passwd $uid); then
  echo "User $name exists!"
  [ "$pwline" == "$PWLINE" ] || exit 101;
else
  adduser --system --home $homedir --shell $shell --uid $uid --group --
gecos $name $name
fi
adduser $name kvm
adduser $name libvirtd

# Mount homedir
mount -t nfs $nfsserver/$name $homedir || exit 201
install $homedir/autoubuntu /etc/init.d/autoubuntu

#Configure Libvirt
grep $name /etc/libvirt/libvirtd.conf ||
     mv /etc/libvirt/libvirtd.conf /etc/libvirt/libvirtd.conf.old
install $homedir/libvirtd.conf /etc/libvirt/
grep $name /etc/libvirt/qemu.conf ||
     mv /etc/libvirt/qemu.conf /etc/libvirt/qemu.conf.old
install $homedir/qemu.conf /etc/libvirt/

umount $homedir

update-rc.d libvirt-bin defaults
service libvirt-bin restart

#Configure NFS
install -d -o $name -g $name /opt/opennebula /opt/opennebula_shared

# Install startup/reboot/shutdown script
chmod +x  /etc/init.d/autoubuntu
update-rc.d autoubuntu defaults
service autoubuntu start
```

autoubuntu.sh

```
   #! /bin/bash
/etc/init.d/autoubuntu
   #

   ### BEGIN INIT INFO
   # Provides:          autoubuntu
   # Required-Start:    $remote_fs $syslog
   # Required-Stop:     $remote_fs $syslog
   # Default-Start:     2 3 4 5
   # Default-Stop:      0 1 6
   # Short-Description: Start daemon at boot time
```

```
# Description:       Enable service provided by autoubuntu.
### END INIT INFO

# Some things that run always
touch /var/lock/autoubuntu
homedir="/home/oneadmin"
name="oneadmin"
nfsserver="fas2-
external.storage.ua.pt:/vol/F2_aggr_SATA_IEETA0_vol_CloudPT/CloudPT/2Labs"
oneserver="cloudpt2.clients.ua.pt"
DNS="193.136.172.21"

# Carry out specific functions when asked to by the system
case "$1" in
  start)
    echo "Starting script autoubuntu "
    TIME=$(date)
    echo "Starting $TIME"

    # check if has network, otherwise do not execute
    if ping -c 1 $DNS &> /dev/null
    then

        # mount NFS directories
        mount -o vers=3 -t nfs $nfsserver/one-var  /opt/opennebula_shared
        mount -o vers=3 -t nfs $nfsserver/$name $homedir
        ln -s /opt/opennebula_shared /opt/opennebula/var

        if [ ! -f $homedir/stop ]  # stop script if something is wrong
        then

            # add host to OpenNebula and Campus cluster
            HNAME=$(hostname)
            NEEDLE="not found"
            STRING=$(ssh $name@$oneserver "onehost show $HNAME")

            if [[ "$STRING" == *"$NEEDLE"* ]] # Host does not exist
            then
              ssh $name@$oneserver "onehost create $HNAME --im kvm --vm kvm
--net ebtables"
              ssh $name@$oneserver "onecluster addhost campus $HNAME"
             fi

            ssh $name@$oneserver "onehost enable $HNAME"

            # configure bridge
            DEVICENAME=$(ip -o link show | awk '{print $2,$9}' |grep ': UP'
| cut -d ':' -f 1)
            IP=$(/sbin/ifconfig $DEVICENAME | sed -n '/inet / { s# *inet
addr:##; s# .*##p }')
            BRIDGE="braulas0"

            declare -a IPARR
            IPARR=(`echo ${IP//./ }`)
            GATEWAY=$(echo ${IPARR[0]}.${IPARR[1]}.${IPARR[2]}.254)

            if [[ "$DEVICENAME" != "$BRIDGE" ]] # Bridge does not exist
            then
                        brctl addbr $BRIDGE
                        brctl addif $BRIDGE $DEVICENAME
                        ifconfig $DEVICENAME 0.0.0.0
                        ifconfig $BRIDGE $IP
                        ifconfig $BRIDGE up
```

```
                        ip route add default via $GATEWAY dev $BRIDGE
                        brctl stp $BRIDGE off
                fi
            fi
        fi
    ;;
      stop)
        echo "Stopping script autoubuntu"
            TIME=$(date)
        echo "Stopping $TIME"
            # umount NFS directories
            umount /opt/opennebula_shared
            rm /opt/opennebula/var
            umount /opt/opennebula
            umount $homedir
            # disable OpenNebula host
            HNAME=$(hostname)
            ssh $name@$oneserver "onehost disable $HNAME"
        ;;
      *)
        echo "Usage: /etc/init.d/autoubuntu {start|stop}"
        exit 1
        ;;
    esac

    exit 0
```

6. UDDI code

uddiservice.sh

```
#! /bin/bash
# chkconfig: 2345 90 12
# /etc/init.d/uddiservice
#

### BEGIN INIT INFO
# Provides:          uddiservice
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start daemon at boot time
# Description:       Enable service provided by uddiservice.
### END INIT INFO

start() {
echo "Starting script uddiservice "
     TIME=$(date)
     echo "Starting $TIME"  >> /root/log.txt
       touch /var/lock/uddiservice
     cd /root/simple-publish-portable/
       /usr/local/maven/bin/mvn -Pdemo test
}
# Restart the service FOO
stop() {
       echo "Stopping script uddiservice"
         TIME=$(date)
     echo "Stopping $TIME"    >> /root/log.txt
     cd /root/simple-delete/
       /usr/local/maven/bin/mvn -Pdemo test
```

```
        rm -f /var/lock/uddiservice
}

# Carry out specific functions when asked to by the system
case "$1" in
  start)
    start
;;
  stop)
    stop
    ;;
  restart)
    stop
      start
    ;;
  *)
    echo "Usage: /etc/init.d/uddiservice {start|stop}"
    exit 1
    ;;
esac

exit 0
```

SimpleDelete.java

```
package org.apache.juddi.example.publish;

import org.uddi.api_v3.*;
import org.apache.juddi.api_v3.*;
import org.apache.juddi.v3.client.config.UDDIClient;
import org.apache.juddi.v3.client.transport.Transport;
import org.uddi.v3_service.UDDISecurityPortType;
import org.uddi.v3_service.UDDIPublicationPortType;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.net.*;

import org.apache.juddi.v3.client.UDDIConstants;
import org.apache.juddi.v3.client.config.UDDIClerk;
import java.util.Iterator;
import java.util.Set;
import org.uddi.v3_service.UDDIInquiryPortType;

/**
 * This shows you to interact with a UDDI server by publishing a Business,
 * Service and Binding Template. It uses some fairly generic code that should
be
 * mostly portable to any other UDDI client library and is therefore consider
 * "portable". URLs are set in uddi.xml
 *
 */
public class SimpleDelete {

      private static UDDISecurityPortType security = null;
      private static UDDIPublicationPortType publish = null;
      private TModel keygen = null;
      private static UDDIInquiryPortType inquiry = null;
      private static UDDIClient client = null;
      private static UDDIClerk clerk = null;
      private String domain_prefix = "uddi:juddi.apache.org:";
      private String lang = ""; //"en";
```

```
      private String myServiceName = "Calculator";
      private String myBusinessKey = "uddi:juddi.apache.org:2ad5499d-dc4e-
45c9-beb6-0f66849ee38f";
      private String myAccessPointApp = ":8080/HelloWorld-20141111/?wsdl";

      public SimpleDelete()
      {
            try {
            // create a client and read the config in the archive;
            // you can use your config file name
            UDDIClient uddiClient = new UDDIClient("META-INF/uddi.xml");
            // a UddiClient can be a client to multiple UDDI nodes, so
            // supply the nodeName (defined in your uddi.xml.
            // The transport can be WS, inVM, RMI etc which is defined in the
uddi.xml
            Transport transport = uddiClient.getTransport("default");
            // Now you create a reference to the UDDI API
            security = transport.getUDDISecurityService();
            publish = transport.getUDDIPublishService();
            } catch (Exception e) { e.printStackTrace(); }
      }

      private String ServiceLookUpByName()
      {
      String myServiceID = "";
      try{
      FindService fs = new FindService();
      fs.setFindQualifiers(new FindQualifiers());
      fs.getFindQualifiers().getFindQualifier().
add(UDDIConstants.EXACT_MATCH);
      fs.getName().add(new Name(myServiceName, lang));
      ServiceList findService = inquiry.findService(fs);

      GetServiceDetail gsd = new GetServiceDetail();
      for (int i = 0; i <
findService.getServiceInfos().getServiceInfo().size(); i++) {

      gsd.getServiceKey().add(findService.getServiceInfos().getServiceInfo().g
et(i).getServiceKey());
                  myServiceID =
findService.getServiceInfos().getServiceInfo().get(i).getServiceKey();
                  System.out.println("!!! " +
findService.getServiceInfos().getServiceInfo().get(i).getServiceKey() );
            }

            }catch(Exception exe){return "";}
            return myServiceID;
      }

    private void Init()
      {
        try {
            client = new UDDIClient("META-INF/uddi.xml");
            clerk = client.getClerk("default");
            Transport transport = client.getTransport("default");
            security = transport.getUDDISecurityService();
            inquiry = transport.getUDDIInquiryService();
            publish = transport.getUDDIPublishService();
        } catch (Exception e) { e.printStackTrace();}

    }

      public void deleteService(String authInfo, String serviceKey)
```

```
            {
                  try {
                  DeleteService ds = new DeleteService();
                  ds.setAuthInfo(authInfo);
                  ds.getServiceKey().add(serviceKey);
                  publish.deleteService(ds);
                  } catch (Exception e) { System.out.println("#### Exception:
" + e);}
            }

      private AuthToken authenticateMe()
      {
            AuthToken myPubAuthToken = new AuthToken();
            try{
            GetAuthToken getAuthTokenMyPub = new GetAuthToken();
            getAuthTokenMyPub.setUserID("uddiadmin");
            getAuthTokenMyPub.setCred("password");
            myPubAuthToken = security.getAuthToken(getAuthTokenMyPub);
            }
            catch (Exception e) {}
            return myPubAuthToken;
      }

      public static void main(String args[]) {
                  SimpleDelete sp = new SimpleDelete();
                  sp.Init();
                  String myServiceID = sp.ServiceLookUpByName();

      sp.deleteService(sp.authenticateMe().getAuthInfo(),myServiceID);
      }
}
```

SimplePublishPortable.java

```
package org.apache.juddi.example.publish;

import org.uddi.api_v3.*;
import org.apache.juddi.api_v3.*;
import org.apache.juddi.v3.client.config.UDDIClient;
import org.apache.juddi.v3.client.transport.Transport;
import org.uddi.v3_service.UDDISecurityPortType;
import org.uddi.v3_service.UDDIPublicationPortType;
import java.io.IOException;
import java.net.*;

import java.util.*;
import org.apache.juddi.v3.client.UDDIConstants;
import org.apache.juddi.v3.client.config.UDDIClerk;
import java.util.Iterator;
import java.util.Set;
import org.uddi.v3_service.UDDIInquiryPortType;

/**
* This shows you to interact with a UDDI server by publishing a Business,
* Service and Binding Template. It uses some fairly generic code that should
be
* mostly portable to any other UDDI client library and is therefore consider
* "portable". URLs are set in uddi.xml
*
*/

public class SimplePublishPortable {
```

```
	private static UDDISecurityPortType security = null;
	private static UDDIPublicationPortType publish = null;
	private static TModel keygen = null;
	private String domain_prefix = "uddi:juddi.apache.org:";
	private String lang = ""; //"en";
	private String myServiceName = "Calculator";
	private String myBusinessKey = "uddi:juddi.apache.org:2ad5499d-dc4e-
45c9-beb6-0f66849ee38f";
	private String myAccessPointApp = ":8080/HelloWorld-
20141111/hello?wsdl";

	public SimplePublishPortable() {
		try {
		// create a client and read the config in the archive;
		// you can use your config file name
		UDDIClient uddiClient = new UDDIClient("META-INF/uddi.xml");
		// a UddiClient can be a client to multiple UDDI nodes, so
		// supply the nodeName (defined in your uddi.xml.
		// The transport can be WS, inVM, RMI etc which is defined in the
uddi.xml
		Transport transport = uddiClient.getTransport("default");
		// Now you create a reference to the UDDI API
		security = transport.getUDDISecurityService();
		publish = transport.getUDDIPublishService();
		} catch (Exception e) {
				e.printStackTrace();
		} catch (Exception e) {
				e.printStackTrace();
		}
	}

	/**
	 * This function shows you how to publish to UDDI using a fairly generic
	 * mechanism that should be portable (meaning use any UDDI v3 library
	 */
	public void publish() {
	try {
		// Login aka retrieve its authentication token
		GetAuthToken getAuthTokenMyPub = new GetAuthToken();
		getAuthTokenMyPub.setUserID("uddiadmin");
		getAuthTokenMyPub.setCred("password");
		AuthToken myPubAuthToken =
security.getAuthToken(getAuthTokenMyPub);
		System.out.println(getAuthTokenMyPub.getUserID() + "'s AUTHTOKEN =
" + "******* never log auth tokens!");

		// Creating the parent business entity
		BusinessEntity myBusEntity = new BusinessEntity();
		Name myBusName = new Name();
		myBusName.setValue("CloudCampus");
		myBusEntity.getName().add(myBusName);

		// Binding to a service to save
		BusinessService myService = new BusinessService();
		myService.setBusinessKey(myBusinessKey);
		Name myServName = new Name();
		myServName.setValue(myServiceName);
	  myServName.setValue(myServiceName);
		myService.getName().add(myServName);

		// Add binding templates, etc...
		BindingTemplate myBindingTemplate = new BindingTemplate();
		AccessPoint accessPoint = new AccessPoint();
```

```
      accessPoint.setUseType(AccessPointType.WSDL_DEPLOYMENT.toString());
            accessPoint.setValue("http://" + getIPAddress() +
myAccessPointApp);
            myBindingTemplate.setAccessPoint(accessPoint);
            BindingTemplates myBindingTemplates = new BindingTemplates();
            //optional but recommended step
            myBindingTemplate = UDDIClient.addSOAPtModels(myBindingTemplate);
            myBindingTemplates.getBindingTemplate().add(myBindingTemplate);

            myService.setBindingTemplates(myBindingTemplates);

            // Adding the service to the "save" structure
            SaveService ss = new SaveService();
            ss.getBusinessService().add(myService);
            ss.setAuthInfo(myPubAuthToken.getAuthInfo());
            ServiceDetail sd = publish.saveService(ss);
            String myServKey = sd.getBusinessService().get(0).getServiceKey();
            System.out.println("myService key:  " + myServKey);

            security.discardAuthToken(new
DiscardAuthToken(myPubAuthToken.getAuthInfo()));
            // published a business and service via the jUDDI API!
            System.out.println("Success!");

            } catch (Exception e) {
                        e.printStackTrace();
            }
      }

      private String ServiceLookUpByName()
      {
            String myServiceID = "";
            try{
            FindService fs = new FindService();
            fs.setFindQualifiers(new FindQualifiers());

      fs.getFindQualifiers().getFindQualifier().add(UDDIConstants.EXACT_MATCH)
;
            fs.getName().add(new Name(myServiceName, lang));
            ServiceList findService = inquiry.findService(fs);

            GetServiceDetail gsd = new GetServiceDetail();
            for (int i = 0; i <
findService.getServiceInfos().getServiceInfo().size(); i++) {

      gsd.getServiceKey().add(findService.getServiceInfos().getServiceInfo().g
et(i).getServiceKey());
                  myServiceID =
findService.getServiceInfos().getServiceInfo().get(i).getServiceKey();
                  System.out.println("!!! " +
findService.getServiceInfos().getServiceInfo().get(i).getServiceKey() );
            }
            }catch(Exception exe){
            //Your error handling code
                  return "";
            }
            return myServiceID;
      }

      private AuthToken authenticateMe()
      {
            AuthToken myPubAuthToken = new AuthToken();
```

```
            try{
                  GetAuthToken getAuthTokenMyPub = new GetAuthToken();
                  getAuthTokenMyPub.setUserID("uddiadmin");
                  getAuthTokenMyPub.setCred("password");
                  myPubAuthToken = security.getAuthToken(getAuthTokenMyPub);
            }
            catch (Exception e) {}
            return myPubAuthToken;
      }

      public static String getIPAddress()
      {
        String ethAddress = "127.0.0.1";
            try{
            for (Enumeration<NetworkInterface> ifaces =
                     NetworkInterface.getNetworkInterfaces();
                   ifaces.hasMoreElements(); )
            {
                  NetworkInterface iface = ifaces.nextElement();
                  for (Enumeration<InetAddress> addresses =
                           iface.getInetAddresses();
                         addresses.hasMoreElements(); )
                  {
                        InetAddress address = addresses.nextElement();
                        if (iface.getName().toString().contains("eth0"))
                                    ethAddress =
address.toString().substring(1);
                  }
            }
            }
            catch(Exception ex){}
            return ethAddress;
      }

      public static void main(String args[]) {
                  SimplePublishPortable sp = new SimplePublishPortable();
                  String myServiceID = sp.ServiceLookUpByName();
                  sp.publish();
      }
}
```

B. OpenNebula code

1. Scheduler

SchedulerTemplate.cc

```
#include "SchedulerTemplate.h"

const char * SchedulerTemplate::conf_name="sched.conf";

void SchedulerTemplate::set_conf_default()
{
    SingleAttribute *   attribute;
    VectorAttribute *   vattribute;
    string              value;
    map<string,string>  vvalue;

/*#*****************************************************************************
# Daemon configuration attributes
#-------------------------------------------------------------------------------
#  XML_RPC_MESSAGE_SIZE
#  ONED_PORT
#  SCHED_INTERVAL
#  MAX_VM
#  MAX_DISPATCH
#  MAX_HOST
#  DEFAULT_SCHED
#  DEFAULT_DS_SCHED
#  LIVE_RESCHEDS
#  LOG
#-------------------------------------------------------------------------------*/
    // XML_RPC_MESSAGE_SIZE
    value = "1073741824";

    attribute = new SingleAttribute("MESSAGE_SIZE",value);
    conf_default.insert(make_pair(attribute->name(),attribute));

    // ONED_PORT
    value = "2633";

    attribute = new SingleAttribute("ONED_PORT",value);
    conf_default.insert(make_pair(attribute->name(),attribute));

    // SCHED_INTERVAL
    value = "30";

    attribute = new SingleAttribute("SCHED_INTERVAL",value);
    conf_default.insert(make_pair(attribute->name(),attribute));

    // MAX_VM
    value = "300";

    attribute = new SingleAttribute("MAX_VM",value);
    conf_default.insert(make_pair(attribute->name(),attribute));

    // MAX_DISPATCH
    value = "30";

    attribute = new SingleAttribute("MAX_DISPATCH",value);
    conf_default.insert(make_pair(attribute->name(),attribute));
```

```
    //MAX_HOST
    value = "1";

    attribute = new SingleAttribute("MAX_HOST",value);
    conf_default.insert(make_pair(attribute->name(),attribute));

    //LIVE_RESCHEDS
    value = "0";

    attribute = new SingleAttribute("LIVE_RESCHEDS",value);
    conf_default.insert(make_pair(attribute->name(),attribute));

    //DEFAULT_SCHED
    vvalue.clear();
    vvalue.insert(make_pair("POLICY","1"));

    vattribute = new VectorAttribute("DEFAULT_SCHED",vvalue);
    conf_default.insert(make_pair(vattribute->name(),vattribute));

    //DEFAULT_DS_SCHED
    vvalue.clear();
    vvalue.insert(make_pair("POLICY","1"));

    vattribute = new VectorAttribute("DEFAULT_DS_SCHED",vvalue);
    conf_default.insert(make_pair(vattribute->name(),vattribute));

    //LOG CONFIGURATION
    vvalue.clear();
    vvalue.insert(make_pair("SYSTEM","file"));
    vvalue.insert(make_pair("DEBUG_LEVEL","3"));

    vattribute = new VectorAttribute("LOG",vvalue);
    conf_default.insert(make_pair(vattribute->name(),vattribute));
}

string SchedulerTemplate::get_policy() const
{
    int    policy;
    string rank;
    istringstream iss;

    vector<const Attribute *> vsched;
    const  VectorAttribute *  sched;

    get("DEFAULT_SCHED", vsched);

    if (vsched.empty())
    {
        return "";
    }

    sched = static_cast<const VectorAttribute *> (vsched[0]);
    iss.str(sched->vector_value("POLICY"));
    iss >> policy;

    switch (policy)
    {
        case 0: //Packing
            rank = "RUNNING_VMS";
        break;

        case 1: //Striping
            rank = "- RUNNING_VMS";
```

```
        break;

        case 2: //Load-aware
            rank = "FREE_CPU";
        break;

        case 3: //Custom for CloudCampus
            rank = sched->vector_value("RANK");
        break;

        case 4: //Fixed
            rank = "PRIORITY";
        break;

        default:
            rank = "";
    }
    return rank;
}

string SchedulerTemplate::get_ds_policy() const
{
    int    policy;
    string rank;
    istringstream iss;
    vector<const Attribute *> vsched;
    const  VectorAttribute *  sched;

    get("DEFAULT_DS_SCHED", vsched);

    if (vsched.empty())
    {
        return "";
    }

    sched = static_cast<const VectorAttribute *> (vsched[0]);
    iss.str(sched->vector_value("POLICY"));
    iss >> policy;

    switch (policy)
    {
        case 0: //Packing
            rank = "- FREE_MB";
        break;

        case 1: //Striping
            rank = "FREE_MB";
        break;

        case 2: //Custom for CloudCampus
            rank = sched->vector_value("RANK");
        break;

        case 3: //Fixed
            rank = "PRIORITY";
        break;

        default:
            rank = "";
    }
    return rank;
}
```

2. Sunstone web interface

vms-tab.js

```
[…]

/********************** Haizea scheduling **********************/

        var haizeaStart = "best_effort"; // "best_effort", "now",
"+00:00:30" or "2013-01-31 11:00:00"

        if ($("input[name='vm_datetime']:checked").val() == "schedule")
            haizeaStart = $("#job_starttime").val();
        else
            haizeaStart = $("input[name='vm_datetime']:checked").val();

        var haizeaDuration = "01:00:00"; // unlimited or "01:00:00"

        if ($("#job_duration").val().length > 0 &&
$("#job_duration").val().trim() != "")
            haizeaDuration = $("#job_duration").val();
        else
            haizeaDuration = "unlimited";

        var haizeaPreemptible = "no"; // yes or no

        var haizeaContent = '\nHAIZEA = [start = "' + haizeaStart + '",\n
duration = "' + haizeaDuration + '",\n preemptible = "' + haizeaPreemptible
+ '"]';

        /********************** Haizea scheduling **********************/

        // !!!!change template and instantiate
        var nextNumberTemplate =
parseInt($('#template_id2')[0][$('#template_id2')[0].length - 1].value) + 1;

        // Get and update template text
        var replaceTemplateId =
document.getElementById("template_template_update_textarea").value.substring
(document.getElementById("template_template_update_textarea").value.search("
\nTEMPLATE_ID=\"?\""))
        document.getElementById("template_template_update_textarea").value =
document.getElementById("template_template_update_textarea").value.replace(r
eplaceTemplateId, "\nTEMPLATE_ID=\"" + nextNumberTemplate + "\"")
        var replaceName =
document.getElementById("template_template_update_textarea").value.substring
(document.getElementById("template_template_update_textarea").value.search("
\nNAME=\"?\""))
        document.getElementById("template_template_update_textarea").value =
document.getElementById("template_template_update_textarea").value.replace(r
eplaceName, "\nNAME=\"JOB_" + document.getElementById("job_name").value +
"\"")

        template_text = $("#template_template_update_textarea").val() +
haizeaContent;
        document.getElementById("template_textarea").value = template_text;
        document.getElementById("template_template_update_textarea").value =
template_text;
        document.getElementById("textarea_vm_template").value =
template_text;

        // Create template
```

```
        var template = $('textarea#textarea_vm_template',
$create_template_dialog).val();

        // Wrap it in the "vm" object and create
        template = { "vmtemplate": { "template_raw": template} };
        Sunstone.runAction("Template.create", template);

        if (vm_name.indexOf("%i") == -1) { //no wildcard
            for (var i = 0; i < n_times_int; i++) {
                // Instantiate new template
                Sunstone.runAction("Template.instantiate",
nextNumberTemplate, vm_name);
            };
        }
        else { //wildcard present: replace wildcard
            var name = "";

            for (var i = 0; i < n_times_int; i++) {
                // Instantiate new template
                name = vm_name.replace(/%i/gi, i);
                Sunstone.runAction("Template.instantiate",
nextNumberTemplate, name);
            };
        };

        setTimeout(function () {
            Sunstone.runAction("VM.list");
        }, 1500);
        $create_job_dialog.dialog('close');
        return false;
    });

}

    /* new job */
    setupCreateJobDialog();
    /* new job */

    setupVMTemplateUpdateDialog();
    setVMAutorefresh();
    setupVNC();
    hotpluggingOps();

    initCheckAllBoxes(dataTable_vMachines);
    tableCheckboxesListener(dataTable_vMachines);
    infoListener(dataTable_vMachines, 'VM.showinfo');

    $('div#vms_tab div.legend_div').hide();

    /* new job - slider*/
    $('#smart-slider').strackbar({ callback: onTick, sliderHeight: 4, style:
'style1', animate: false, ticks: false, labels: true, trackerHeight: 20,
trackerWidth: 19, maxValue: 10 });
    var datenow = new Date().toISOString().replace("T", " ").substring(0,
19);
    document.getElementById("job_starttime").value = datenow;
    document.getElementById("job_endtime").value = datenow.substring(0, 12)
+ (parseInt(datenow[12]) + 1).toString() + datenow.substring(13, 20);
    /* end new job - slider*/

})
```

C. VM Templates

1. Windows Server 2008 template

```
RAW=[TYPE="kvm"]
SCHED_REQUIREMENTS="CLUSTER = \"ieeta\""
NIC=[MODEL="e1000",NETWORK="Aulas Network",NETWORK_UNAME="oneadmin"]
FEATURES=[ACPI="yes"]
GRAPHICS=[LISTEN="0.0.0.0",TYPE="vnc",KEYMAP="pt"]
CPU="0.5"
MEMORY="2048"
DISK=[IMAGE_UNAME="oneadmin",IMAGE="Windows Server 2008 SQL VS2010"]
OS=[ARCH="x86_64",BOOT="hd"]
NAME="Windows2008-SQL08-VS10"
TEMPLATE_ID="0"
CONTEXT=[TOKEN="YES",HOSTNAME="Win2008-$VMID", USERNAME="admin",
IP_PUBLIC="$NIC[IP, NETWORK=\"Aulas Network\"]",
GATEWAY="192.168.160.254",
PASSWORD="Password!23",NAGIOS_IP="192.168.160.111",
FILES="/opt/opennebula/var/context-scripts/windows/updateNagios.vbs
/opt/opennebula/var/context-scripts/windows/curl.exe
/opt/opennebula/var/context-scripts/windows/Shutdown.cmd
/opt/opennebula/var/context-scripts/windows/startup.vbs
/opt/opennebula/var/context-scripts/windows/one-context.ps1
/opt/opennebula/var/context-scripts/windows/SetupComplete.cmd
/opt/opennebula/var/context-scripts/windows/unattend.xml
/opt/opennebula/var/context-scripts/windows/README.txt"]
VCPU="2"
```

2. Ubuntu Server 12.04 template

```
RAW=[TYPE="kvm"]
NIC=[NETWORK="Aulas Network",NETWORK_UNAME="oneadmin"]
FEATURES=[ACPI="yes"]
GRAPHICS=[LISTEN="0.0.0.0",TYPE="vnc",KEYMAP="pt"]
CPU="0.5"
MEMORY="1024"
DISK=[IMAGE_UNAME="oneadmin",DRIVER="qcow2",IMAGE="Ubuntu Server 14.04
64-bit"]
OS=[ARCH="x86_64",BOOT="hd"]
CONTEXT=[ROOT_PW="Password!23", TOKEN="YES", USER_PUBKEY="id_rsa.pub",
HOSTNAME="Ubuntu12-$VMID", USER_PW="Password!23", USERNAME="oneadmin",
ROOT_PUBKEY="id_rsa.pub",IP_PUBLIC="$NIC[IP, NETWORK=\"Aulas Network\"]",
GATEWAY="192.168.160.254", PASSWORD="Password!23",
NAGIOS_IP="192.168.160.111", NAGIOS_PORT="5000",
FILES="/home/oneadmin/.ssh/id_rsa.pub /opt/opennebula/var/context-
scripts/ubuntu/init.sh /opt/opennebula/var/context-
scripts/ubuntu/onshutdown /etc/resolv.conf"]
SCRIPT="apt-get install curl nagios-nrpe-server nagios-nrpe-plugin -f"
```

D. Federated authentication

1. fpa2-ieeta-medatada.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor
      xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:shibmd="urn:mace:shibboleth:metadata:1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      entityID="https://fpa2.ieeta.pt">

  <md:Extensions xmlns:alg="urn:oasis:names:tc:SAML:metadata:algsupport">
    <alg:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha512"/>
    <alg:DigestMethod Algorithm="http://www.w3.org/2001/04/xmldsig-
more#sha384"/>
    <alg:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
    <alg:DigestMethod Algorithm="http://www.w3.org/2001/04/xmldsig-
more#sha224"/>
    <alg:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <alg:SigningMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
sha512"/>
    <alg:SigningMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
sha384"/>
    <alg:SigningMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
sha256"/>
    <alg:SigningMethod Algorithm="http://www.w3.org/2009/xmldsig11#dsa-
sha256"/>
    <alg:SigningMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"/>
    <alg:SigningMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-
sha1"/>
  </md:Extensions>
    <md:SPSSODescriptor
protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol
urn:oasis:names:tc:SAML:1.1:protocol
http://schemas.xmlsoap.org/ws/2003/07/secext"
xmlns="urn:oasis:names:tc:SAML:2.0:metadata">
    <md:KeyDescriptor>
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:X509Data>
          <ds:X509SubjectName>CN=fpa2.ieeta.pt,O=FPA2 Service
Provider,L=Aveiro,ST=Aveiro,C=PT</ds:X509SubjectName>

<ds:X509Certificate>MIIC7jCCAdagAwIBAgIJAP3DbuKKzwJvMA0GCSqGSIb3DQEBBQUAMBgxFj
AUBgNV
BAMTDWZwYTIuaWVldGEucHQwHhcNMTMxMTI1MTQ1NjEwWhcNMjMxMTIzMTQ1NjEw
WjAYMRYwFAYDVQQDEw1mcGEyLmllZXRhLnB0MIIBIjANBgkqhkiG9w0BAQEFAAOC
AQ8AMIIBCgKCAQEAoc8813XpqRcwBWLlJPWYgtPbqLdLVYfxrcr87ewsqPdIGDLc
glAQaa6Z6C94rNzxHmB5au9jVLiLBMohRH7psl7dxK88zhy9Oy1xnDhgBGgzWJhd
C1hkWBDCuMnD8ZRTadT6yq86kF7M/vZA52WKgL3cHPOh8JDzQiNUitFOI1eg2fkC
/E50sDd58sCWu7m+Gf3pE0Pz+rqYcB9XzCrYncjKtSxlVKDUCt2cHjt5w4Qq124H
hbxsK+avLvZgDl0FcesHnnNLcwRBF9bs9HCLY3N5NWxwzMdMgm87R3aMkZD2Wb3e
gdu3g7U59Bzon1w89YcJukqUhqIpdt/q1OEWwQIDAQABozswOTAYBgNVHREEETAP
gg1mcGEyLmllZXRhLnB0MB0GA1UdDgQWBBRYj8au3p0HDZJJnKlKq+SS+l3U4DAN
BgkqhkiG9w0BAQUFAAOCAQEAU89/ySB1n/U6bNRSX8fnN17MtILPnG/wkB3jGGCc
diyPDPZcXAZkBFmzFfd+7Ar+IRtEGGsvCKCcv2A3m02GQNzvQxTA96K3c9TAVQRZ
/W2fh/HxlYTtCceBtRID275Gm3iVB0WoM9o63h3bZ1IqxcgxpvCx+oz0DaETNvfW
fgv+BRIUzM2oPOt9uPz09T5YsPJ7zO8ldllPnRudk2ltBs8PX2f34hYYqo91FInw
MDQLQiBST72cPNf26xS9AVaSPPEMhNI86EypZ+MsgyQYGGUrNrGaYC+wM0gw5TQ2
c5egAGbH/EVa5v1tcEhb2dKIKiapLXcI62ZAQPlz4GeB8w==
</ds:X509Certificate>
```

```
        </ds:X509Data>
      </ds:KeyInfo>
      <md:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-
cbc"/>
      <md:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes192-
cbc"/>
      <md:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-
cbc"/>
      <md:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
      <md:EncryptionMethod Algorithm="http://www.w3.org/2009/xmlenc11#rsa-
oaep"/>
      <md:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-
oaep-mgf1p"/>
    </md:KeyDescriptor>
      <md:AssertionConsumerService
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
Location="https://fpa2.ieeta.pt/Shibboleth.sso/SAML2/POST" index="0"/>
      <md:AssertionConsumerService
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST-SimpleSign"
Location="https://fpa2.ieeta.pt/Shibboleth.sso/SAML2/POST-SimpleSign"
index="1"/>
      <md:AssertionConsumerService
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact"
Location="https://fpa2.ieeta.pt/Shibboleth.sso/SAML2/Artifact" index="2"/>
      <md:AssertionConsumerService
Binding="urn:oasis:names:tc:SAML:1.0:profiles:artifact-01"
Location="https://fpa2.ieeta.pt/Shibboleth.sso/SAML2/Artifact" index="3"/>
      <md:AssertionConsumerService
Binding="urn:oasis:names:tc:SAML:1.0:profiles:browser-post"
Location="https://fpa2.ieeta.pt/Shibboleth.sso/SAML/POST" index="4"/>
  </md:SPSSODescriptor>
<md:Organization xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata">
            <md:OrganizationName xml:lang="en">IEETA</md:OrganizationName>
            <md:OrganizationDisplayName xml:lang="en"> Institute of
Electronics and Telematics Engineering of Aveiro</md:OrganizationDisplayName>
            <md:OrganizationURL
xml:lang="en">http://www.ieeta.pt</md:OrganizationURL>
        </md:Organization>
        <md:ContactPerson contactType="technical"
xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata">
            <md:GivenName>Andre</md:GivenName>
            <md:SurName>Monteiro</md:SurName>
            <md:EmailAddress>andremonteiro@ua.pt</md:EmailAddress>
        </md:ContactPerson>
   </md:EntityDescriptor>
```

E. Benchmarks

Table 7.1 - RAM benchmark (MB/s), more is better

cloudpt2 HP Proliant BL460c	lab HP Compaq 8200	Description
11001,81	6190,10	Scale - Float
11529,80	6302,43	Average - Int
11281,25	6314,58	Scale - Int
11398,43	6149,19	Copy - Float
11300,04	6264,86	Average - Float
11300,04	7062,79	Add - Int
9207,81	6278,21	Triad - Int
9597,62	6274,30	Triad - Float
11043,42	7084,91	Add - Float
11043,42	6648,35	Copy - Int

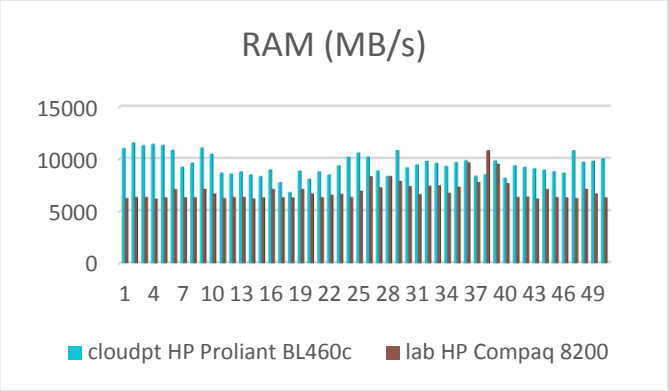

Figure 7.1- RAM benchmark (MB/s)

Table 7.2 - CPU benchmark with Pi test (s), less is better

cloudpt2 HP Proliant BL460c	lab HP Compaq 8200
5,0223650932312	4,7925729751587
5,0722651481628	4,7925729751587
4,7925729751587	4,1757318973541

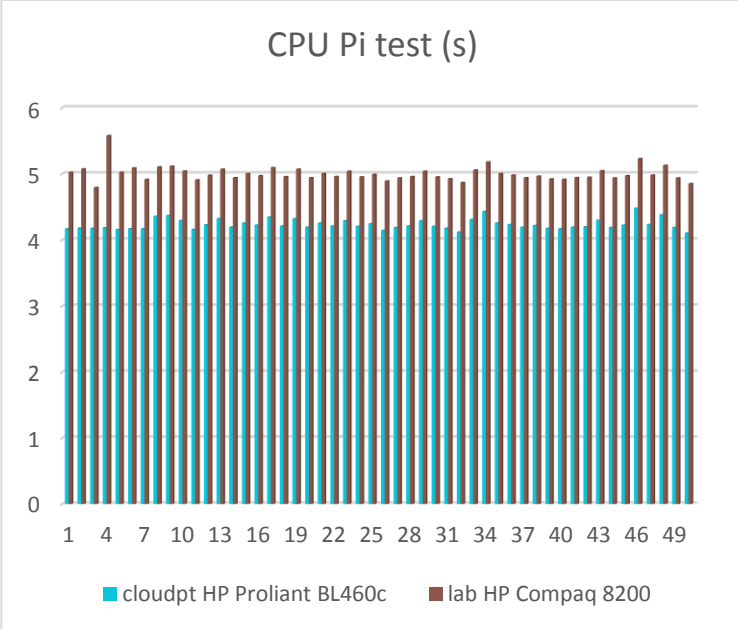

Figure 7.2 - CPU benchmark with Pi test (s)

Table 7.3 - CPU benchmark with C-Ray test (s), less is better

cloudpt2 HP Proliant BL460c	lab HP Compaq 8200
68,34	68,34
68,34	68,34
68,34	68,34

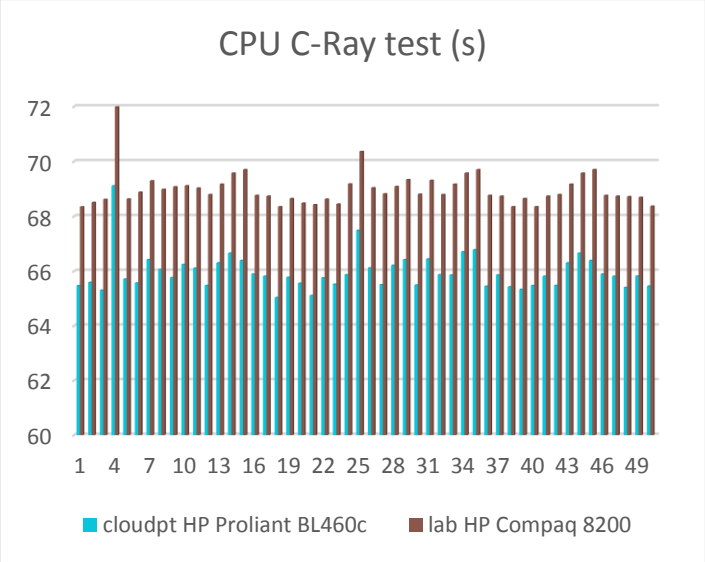

Figure 7.3 - CPU benchmark with C-Ray test (s)

Table 7.4 - CPU benchmark with VPX test (FPS), more is better

cloudpt2 HP Proliant BL460c	lab HP Compaq 8200
18,24	22,34
18,48	22,37
18,80	22,45

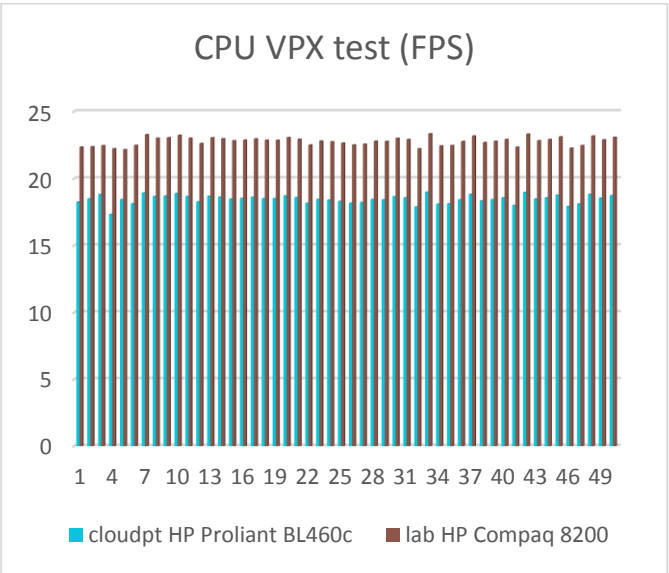

Figure 7.4 - CPU benchmark with VPX test (FPS)

Table 7.5 - CPU benchmark with GCrypt test (s), less is better

cloudpt2 HP Proliant BL460c	lab HP Compaq 8200
2,88	2,64
2,92	2,63
2,88	2,64

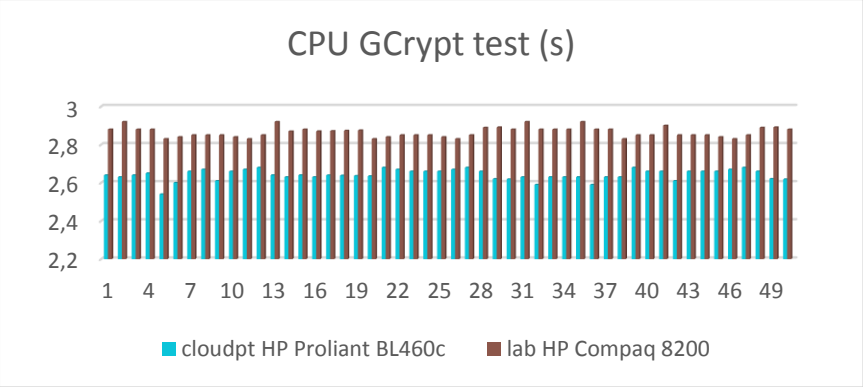

Figure 7.5 - CPU benchmark with GCrypt test (s)

Table 7.6 - CPU benchmark with x264 test (s), less is better

cloudpt2 HP Proliant BL460c	lab HP Compaq 8200
49,17	54,07
48,49	53,82
48,88	54,21
48,26	54,02
47,91	53,90

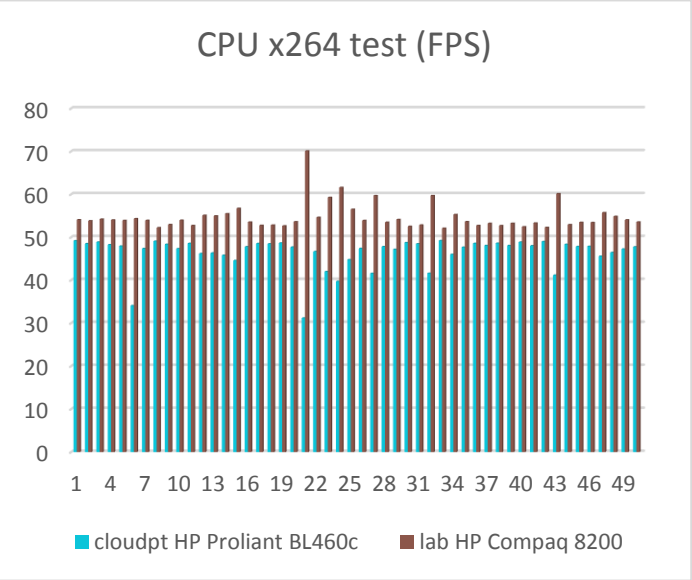

Figure 7.6 - CPU benchmark with x264 test (s)

Table 7.7 – Network benchmark test (s), less is better

cloudpt2 HP Proliant BL460c	lab HP Compaq 8200
21,095597982	17,643713951
20,900372982	17,747543096
21,208943128	17,394306182

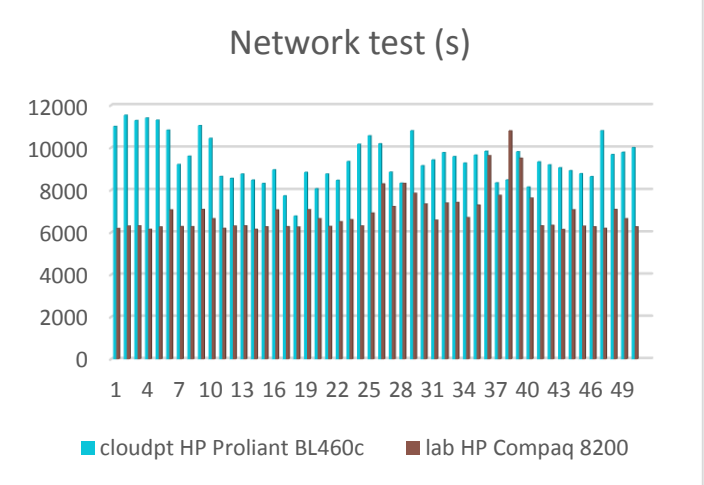

Figure 7.7 - benchmark network test (s)