UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Departamento de Electrónica e Computación

Ph.D. THESIS

PLATFORM AS A SERVICE INTEGRATION FOR SCIENTIFIC COMPUTING USING DIRAC.

A dissertation presented by:

Víctor Manuel Fernández Albor

Supervisor:

Anselmo Tomás Fernández Pena

Ricardo Graciani Díaz

Juan Jose Saborido Silva

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

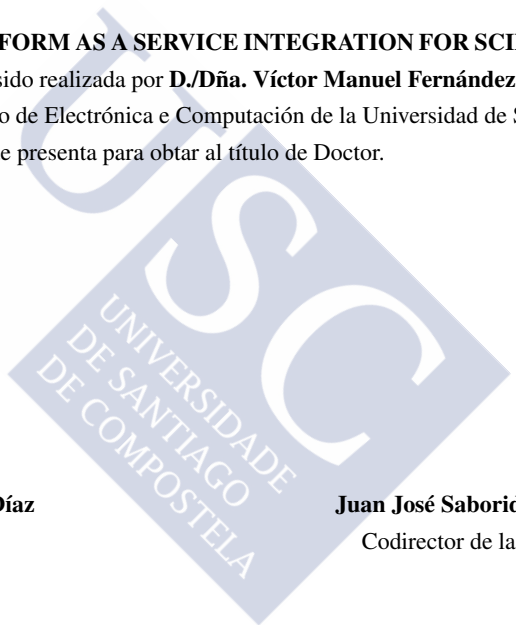
Ricardo Graciani Díaz, Profesor Titular de Universidad del Área de Física Teórica de la Universidad de Barcelona

Juan José Saborido Silva, Profesor Titular de Universidad del Área de Física Atómica Molecular y Nuclear de la Universidad de Santiago de Compostela

Anselmo Tomás Fernández Pena, Profesor Titular de Universidad del Área de Arquitectura y Tecnología de Computadores de la Universidad de Santiago de Compostela

HACEN CONSTAR:

Que la memoria titulada **PLATFORM AS A SERVICE INTEGRATION FOR SCIENTIFIC COMPUTING USING DIRAC.** ha sido realizada por **D./Dña. Víctor Manuel Fernández Albor** bajo nuestra dirección en el Departamento de Electrónica e Computación de la Universidad de Santiago de Compostela, y constituye la Tesis que presenta para obtar al título de Doctor.



Ricardo Graciani Díaz
Codirector tesis

Juan José Saborido Silva
Codirector de la tesis

Anselmo Tomás Fernández Pena
Codirector de la tesis

Doctorando
Autor de la tesis

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

A mi familia, mis padres y mi hermano.

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Contents

List of Figures

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Overview about New Models of Distributed Computing

The term "Distributed computing" is used to describe any sort of operation that involves a distributed system, that is, multiple computers that communicate through a network. In this sense, distributed systems include from tightly connected systems, like a parallel computer or a cluster, to loosely coupled ones, like Grids. A cluster involves a set of nearly identical nodes in close proximity, connected via gigabit or fiber-optical networks [13]. Since the only hardware support for distributed computation in clusters is the network connecting the nodes, all of the application-level parallelism in such environments is achieved via software middleware as Message Passing Interface (MPI).

Grid Computing can be seen as a federation of computer resources from multiple locations. These resources usually are loosely coupled, heterogeneous, and geographically dispersed. The Grid takes its name from an analogy with the electrical "power Grid". The idea is that accessing computer power from a computer Grid would be as simple as accessing electrical power from an electrical Grid.

According to Foster et al. [62], Grid Computing aims to "enable resource sharing and coordinated problem solving in dynamic, multi-institutional virtual organizations". So, Grids provide a distributed computing infrastructure that spans across multiple virtual organizations (VO), where a VO consist of a dynamic set of physically distributed individuals or institutions defined around a set of resource-sharing rules and conditions. The goal of this infrastructure is to enable federated resource sharing in dynamic and distributed environments.

Foster et al. [62] defined the Grid architecture as a provider of protocols and services at five different layers as is shown in the Figure 1.1. At the fabric layer, Grids provide access to different resource types such as compute, storage and network resources, etc. The connectivity layer defines core communication and authentication protocols for easy and secure network transactions. The resource layer defines protocols for the publication, discovery, negotiation, monitoring, accounting and payment of sharing operations on individual resources. The collective layer captures interactions across collections of resources and directory services. Finally, the application layer comprises whatever user applications built on top of the above protocols and APIs and operate in Virtual Organization environments [61].

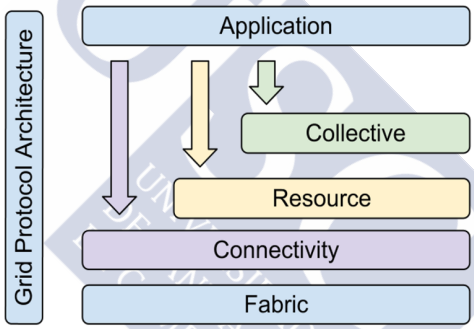


Figure 1.1: Grid protocol architecture.

In the last few years, there has been an evolution that has been a result of a shift in focus from an infrastructure that delivers storage and compute resources, such as Grids, to one that is economy based aiming to deliver more abstract resources and services, as is the case of Cloud Computing. The Cloud Computing consists, basically, in the usage of computing resources that are delivered as a service over a network. The name comes from the use, in system diagrams, of a Cloud-shaped symbol as an abstraction for the complex infrastructure it contains.

There is a little consensus on how to define the Cloud [126] into the world of distributed computing, one of them is the definition of Foster et al. [62]:

"A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet."

There are also multiple versions of definition for Cloud architecture, and in [62] Foster et al. defined a four-layer architecture for Cloud Computing, in comparison to the Grid architecture, composed of fabric, unified resource, platform, and application layers, as is shown in the Figure 1.2.

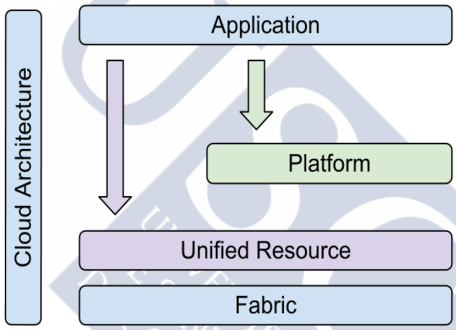


Figure 1.2: Cloud protocol architecture.

In this architecture, the fabric layer contains the raw hardware level resources(computing and network). The unified resource layer contains resources that have been abstracted/encapsulated (usually by virtualization) so that they can be exposed to upper layer and end users as integrated resources, for instance, a virtual computer/cluster, a logical file system, a database system, etc. The platform layer adds on a collection of specialized tools, middleware and services on top of the unified resources to provide a development and/or deployment platform. Finally, the application layer contains the applications that would run in the Clouds.

As for Utility Computing [141], Cloud Computing is not a new paradigm of computing infrastructure; rather, it is a business model in which computing resources, such as computation and storage, are packaged as metered services similar to a physical public utility, such as electricity and public switched telephone networks. Utility computing is typically imple-

mented using other computing infrastructure (e.g. Grids [59]) with additional accounting and monitoring services. It is possible for Clouds to be implemented over existing Grid technologies, leveraging more than a decade of community efforts in standardization, security, resource management, and virtualization support. But, the migration process of applications from one computing environment to another in general, and from Grid to Cloud Computing in particular, is a complex process. It requires a deep understanding of the application to be ported, considering requirements, architecture, functionality etc. to be able to design and restructure the application for the new environment, providing the same or even better functionality. Some major steps have to be considered when porting an application from one computing environment to another [106]:

1. Extract or recover the information from the initial application.
2. Refactor the application to be migrated to the new environment.
3. Assure interoperability between services and data after refactoring.
4. Validate and test the new applications to ensure the same functionalities are provided.

Some lacks of Grid computing systems are related to the use of heterogeneous systems for different user groups. In this way, Cloud Computing offers an innovative vision of resource access with infrastructure flexibility, faster deployment of applications and data, cost control or adaptation of Cloud resources to real needs. There are three basic types of Clouds [97], *Public Clouds*, which are owned and operated by third party service providers, *Private Clouds*, which are those that are built exclusively for an individual enterprise, and *Hybrid Clouds*, which combine the advantages of both the public and private Cloud models. Examples of Public Clouds are Amazon EC2 (Elastic Cloud Computing) [2] with Simple Storage Service (S3) [3], where computing and storage infrastructures are open to public access with a utility pricing model, or Microsoft Azure [136], where as EC2, users rent virtual machines for running their own applications and are charged by the amount of time their virtual machine is running. On the other hand, CloudStack [32] and OpenNebula [103] are examples of Private Clouds, which are open source Cloud implementations, that provide a compatible interface to Amazon's EC2, and allow people to set up a Cloud infrastructure at their premises and experiment with the infrastructure prior to buy commercial services.

In general, Clouds provide services at three different levels (IaaS, PaaS, and Saas [81]), although some providers can choose to expose services at more than one level:

– *Infrastructure as a Service (IaaS)* provisions hardware, software and equipment to deliver software application environments with a resource usage-based pricing model. The infrastructure can scale up and down dynamically based on application resource needs. Some examples are Amazon EC2 or CloudStack.

– *Platform as a Service (PaaS)* offers a high-level integrated environment to build, test and deploy custom applications. Generally, developers will need to accept some restrictions on the type of software they can write in exchange for built-in application scalability. An example is Google's App Engine [70], which enables users to build Web applications on the same scalable systems that power Google applications.

– *Software as a Service (SaaS)* delivers special-purpose software that is remotely accessible by consumers through the Internet. Salesforce is an industry leader in providing online CRM (Customer Relationship Management) Services. In the business model of using software as a service, users are provided access to application software and databases. The Cloud providers manage the infrastructure and platforms on which the applications run. SaaS is sometimes referred to as "on-demand software". SaaS providers generally price applications using a subscription fee. Some of the biggest Cloud Computing services include Web-based email (e.g. Outlook.com, Gmail), social networking (e.g. Facebook, Twitter, LinkedIn), document hosting services, (e.g. Google Docs, Flickr, Picasa, YouTube) and back up services (e.g. Dropbox), etc.

Although Clouds provide services at these three different levels (IaaS, PaaS and Saas), standards for interfaces to these levels still remain to be defined. This leads to interoperability problems between today's Clouds, and there is little business incentives for Cloud providers to invest additional resources in defining and implementing new standards. Network as a Service (NaaS), Storage as a Service (STaaS), Security as a Service (SECaaS), Data as a Service (DaaS), Database as a service (DBaaS), Test Environment as a Service (TEaaS), API as a service (APIaaS), Backend as a service (BaaS) or Desktop virtualization.

1.2 Cloud Computing as an Innovation Process

It is considered an established fact, that the largest computing centres know the benefits of Cloud [9], and they are in a migration process to move their resources from dedicated hosts to

virtualized machines, more flexible and reliable than traditional systems. The Cloud Computing migration involves changes not only in the form that the resources will be available, but also in the internal structure and the software that will be needed by these computing centres. On the other hand, there is Grid software that has been created to provide access to distributed computing systems, in order to allow the researches from different groups run their jobs.

So, the question is: is it possible for Clouds to be implemented over existing Grid technologies? In [62], Foster et al. state that "Cloud Computing not only overlaps with Grid Computing, it is indeed evolved out of Grid Computing an relies on Grid Computing as its backbone and infrastructure support".

Grid Computing was created as a distributed computing system and the main purpose of this technology is to provide the researchers with an easy way to run their jobs. The Grid solved large-scale computation problems using a network of resource-sharing commodity machines that deliver the computation power affordable only by supercomputers and large dedicated clusters at that time. Then, *what advantages exist in the use of Cloud Computing?*, *why the computing centres should do a migration?* Firstly, Cloud Computing can provide flexibility and a reduction in computational costs that the Grid cannot offer. In this sense, the cost reduction will be influenced by the inclusion of virtual machines with the pay-per-use in Public Clouds by user demand. Furthermore, Cloud Computing is the solution for a computing cluster with heterogeneous platforms, that hosts different configurations for different user groups, providing the specific set-up that each different user needs in the cluster. The main motivation behind this can be summarized in two words: "heterogeneity" and "flexibility".

Figure 1.3 shows an overview of the relationship between Clouds and other domains it overlaps with. Supercomputing and Cluster Computing have been more focused on traditional non-service applications. Grid Computing overlaps with all these fields where it is generally considered of lesser scale than supercomputers and Clouds as is shown in the figure 1.3

Summarizing, it is said that Cloud Computing is a new and different way to architect and remotely usage computing resources. Therefore, it is important to study both its shortcomings and its advantages which could help to resolve current Grid issues, as the host dedicated platform for specific groups. Here are some of its significant advantages:

– Lower cost: Cloud Computing requires lower cost for data processing when compared with the older model of maintaining software on local machines. The use of the Cloud removes the need for purchasing software and hardware and shifts the costs to a pay-per-use model.

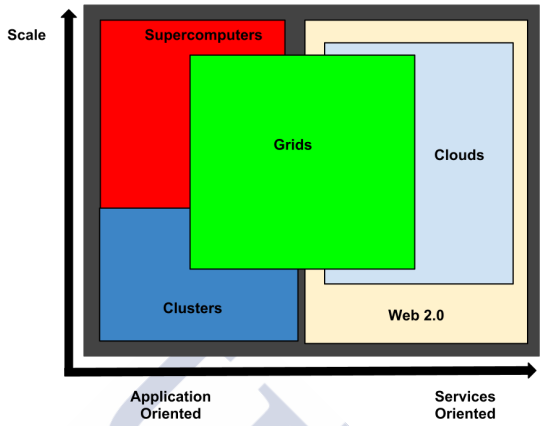


Figure 1.3: Grid an Clouds overview.

– Mobility and flexibility: As the service is delivered through the Internet, users can access the resources no matter where they are located. Instant deployment and the pay-as-you-go business model offer users more flexibility than other computing services. Users do not need to worry about the installation or updates of the applications.

– More capacity: Cloud Computing offers virtually limitless storage and computing power, thus enhancing the user's capability beyond the capacity of their local machines.

– Device independence: Users are no longer tethered to a single computer or network. Existing applications and data follow users through the Cloud.

Thus, the next evolutionary step of High Performance Computing will most likely involve a departure from the mono-platform Grid computing to dynamic and heterogeneous multi-platforms of federated Clouds, connecting together the computational powers of today's clusters with other scientific resources.

1.3 Big Data and MapReduce Paradigm

More than ever, scientific communities need to access large computational resources. Scientists are generating datasets that are increasing exponentially in both complexity and volume, making their analysis a significant challenge. Science is also becoming increasingly global,

with more multinational projects being performed by geographically dispersed scientists that require environments through which they can seamlessly access data, software, and processing resources which usually are distributed and managed under separate administrative domains.

Big Data technologies have arisen as a very viable alternative for problems involving the processing of very large amounts of data. The MapReduce programming model [37] abstracts the common difficulties linked to distributed processing on large clusters, providing an easy-to-use programming model that features automatic parallelization, scalability and data locality-based optimizations. Moreover, the excellent fault tolerance features of popular implementations of MapReduce, like Hadoop [5], make them well-suited for its execution in unstable environments such as commodity clusters or cloud infrastructures.

1.4 Research Objectives

In this PhD thesis, a new approach to the design of distributed systems is explored, focusing in a mix of different Grid, Cloud and Big Data technologies and the provision of a scalable software distribution system. A set of middleware and interware software to seamlessly connect resources and services is also described. Typically, the resources are not only physically apart, but also belong to different organizations which would imply possible administrative barriers. CloudStack [32], OpenNebula [103], DIRAC [40], Hadoop [5] or CernVM-FS [29] constitute the basis of the software solution presented in this thesis.

This PhD thesis proposes a job submission system for several Virtual Organizations [61], which is based in Cloud Computing environments with a central software service repository that provides the necessary software to execute the applications. The proposed mechanism performs resource allocation between non-dedicated and dedicated clusters providing different platforms depending on the user group and with the possibility of sending jobs to different Cloud managers. In other words, Federated Clouds and computer labs of institutions sharing their resources will allow researchers to send their jobs using in each case the most appropriate platform for the software, in collaboration with the Formiga-Cloud project [75]. This PhD thesis also proposes a distributed system as a way to provide the necessary software to run the jobs that can be used by several platforms and that is scalable for the successful mission of this thesis. The proposed mechanism is based on studies of several Cloud managers and efficiency of different software distribution systems.

The main goals of this work are:

– The integration of dedicated and non-dedicated resources, such as computing labs of universities or colleges. This will increase the computing power, particularly at night and weekends, when there is no academic activity.

– A scalable solution for software distribution. This is needed to achieve a low latency and maximize the efficiency in usage of resources.

– An easy management of the computing infrastructure and a seamless tool for job submission by several user communities. This will allow researches to focus on their work rather than worrying about the specific characteristics of the hardware and software of the facilities where the jobs are running.

– Each group of users (Virtual Organization) will have a specific platform, software and user priorities to run their jobs.

– Big Data software integration under Cloud environments for running jobs with distributed datasets.

– Possibility to incorporate further resources when available like Grids, Clouds or Big Data clusters from other providers.

– Inclusion of multi-Cloud Computing managers. The Cloud environment builtin with this system, based in CloudStack, is supposed to be cheap, easy to use and manage. This, together with the use of the DIRAC software are the main components of this thesis. The integration of VMDIRAC, extension of DIRAC, allows the use of other Cloud managers as OpenNebula.

1.5 Thesis Overview

This thesis is organized as follows:

Chapter 2 explains basic features and concepts of the technology on which this PhD thesis is based, focusing on resource management in distributed environments, such as concepts of virtualization and Cloud Computing, Big Data, DIRAC Interware that is the software in charge of job submission, Grid evolution and some history, systems of software distribution, Hight Energy Physics(HEP) software description, and finally federating clouds.

Chapter 3 contains an study about software distribution methods aimed to choose the the most scalable option. Section 3.1 describes some other systems, with benchmarks that were done in these solutions. After that, the deployment, benchmark algorithms and results of test infrastructure are described. In the last part, the final conclusion of the study of software distribution with CernVM-FS is presented, and a new concept of Software Repository Service with the CernVM-FS solution, and the prototype results is shown. Section 3.2 contains the statistical performance analysis of HEP software running in Cloud Technologies.

Chapter 4 describes the methodology, the modelling tools and approaches used throughout the modelling phase in the first part. Second part describes the problem that we are trying to solve in greater detail by analysing the requirements and proposing use-cases. There are four prototypes with use-cases: the first-one as the beginning of the project; the second-one as an evolution applying the learned lessons from the first, the third was created with the feedback provided by previous prototypes and the fourth that allows the Big Data software integration. A potential real world use-case scenario is described, in which the proposed solution may be deployed to efficiently manage the infrastructure and the final goal of the project in a complex computational Cloud environment is defined. The last part, contains the architecture and the design of the three prototypes created for job submission to the infrastructure.

Chapter 5 validates the proposed model. The evaluation is divided into fourth major parts:

- The evaluation and results of CloudStack prototype.
- The evaluation of the Multi-EndPoint Prototype. This section includes the validation of experiments conducted on the deployment of the prototype in a real world use-case with several groups of users, these groups come from the different fields of HEP-Software, and with the feature Multi-SITE[1].
- The evaluation of a new Federated Hybrid Clouds Architecture.
- The evaluation of a Big Data Architecture using the software of Hadoop ecosystem.

Chapter 6 summarizes the results of this research and discusses further enhancements that should be addressed in future work.

[1] "SITE: provider of computing resources in a distributed way."

CHAPTER 2

STATE OF THE ART

This chapter provides a brief overview of thesis technologies, furthermore describes the studies that have relation with the design decisions and the final resulting infrastructure.

2.1 Cloud Computing Overview

The best way to understand the Cloud Computing is to start by server virtualization [12], this concept refers to enable multiple instances of operating systems and applications (virtual machines or VMs) to run on the same physical hardware. The Cloud Computing services are being used with the server virtualization of a cluster or a computer lab. Customers can rent virtual machines running on hardware owned and managed by third-party providers, this is known as **Public Clouds**.

Or instead of that they can virtualize their private infrastructure or cluster, it's called **Private Clouds**. The next section describes virtualization and the basic concepts of Cloud Computing.

2.1.1 Hardware virtualization

Server virtualization is achieved by a hypervisor, which is in many ways similar to that of an operating system. As an operating system manages access by processes to underlying resources, so too a hypervisor must manage access by multiple virtual machines to a single physical machine. In either case the choice of scheduling algorithm will involve a trade-off between factors such as fairness, usage caps and scheduling latency. As in operating systems,

a hypervisor scheduler may be vulnerable to the behaviour of virtual machines, resulting in inaccurate or unfair scheduling. Modern hypervisors may be classified by the methods of executing guest OS code without direct hardware access: (a) binary emulation and translation, (b) para-virtualization, and (c) hardware virtualization support. There are two main open-source hypervisors XEN [139]and Kernel-based Virtual Machine (KVM hereinafter) [90].

XEN has two methods of executing guest OS, the Paravirtualization and the XEN HVM.

The **XEN Paravirtualization** (PV hereinafter) is a form of virtualization, which provides near-native performance and the appearance of isolation at the hardware level, in contrast to another equally popular virtualization platform called *OpenVZ*, which isolates at the OS or process level. The PV runs a complete and unmodified operating system and behaves exactly like a physical machine, excluding the kernel, boot loader and partition layout. By default PV uses the same kernel that the host machine is running, but in XEN case through a technology called *Pygrub* [109], it is also possible to run your own kernel inside your Virtual Private Server (VPS), e.g. the native kernel your favourite linux distribution provides. Table 2.1 shows the advantages and disadvantage of PV.

Paravirtualization advantages	Paravirtualization disadvantages
Complete and secure isolation	Supports Linux only
Runs a full unmodified OS, excluding bootloader + kernel	Limited to only the precreated OS templates
Highly scalable, low overheads	Can not modify the OS options during install
Easy OS reloads with precreated templates	Can not compile and install a custom kernel unless using Pygrub
Direct allocations of RAM + HDD, not oversold and cannot be used by other VPS	Fixed two file system layout with two partitions, one for hdd one for swap
Easily upgraded/downgraded or migrated to another host machine if required	

Table 2.1: Advantages and disadvantage of paravirtualization.

The **XEN Hardware-assisted virtual machine** (HVM) is more popular, and offers the most flexibility. Like PV, XEN HVM provides near-native performance (although the overheads are slightly higher), and provides full hardware isolation, assisted by the Virtualization extensions in modern processors. In a nutshell this means it really is a Virtual Dedicated Server, allowing you to install any operating system, with complete control over every aspect

of it including the kernel, boot loader, file systems + partition layout. Some advantages and disadvantages of HVM are shown in Table 2.2.

HVM advantages	HVM disadvantages
Complete and secure isolation	Slower / more time consuming re-installs
Run Linux, Unix, BSD or Windows	Higher overheads than XEN PV
Customize your kernel, and even your partition layouts	More difficult to upgrade/downgrade or migrate to another host machine if required
Behaves exactly like a real physical server	
Library updated regularly	

Table 2.2: Advantages and disadvantage of Hardware-assisted virtual machine.

KVM is a *full virtualization solution for Linux on x86 hardware* that contains two types of virtualization extensions *(Intel VT and AMD-V)*. It consists of a loadable kernel module, *kvm.ko*, that provides the core virtualization infrastructure and a processor specific module, *kvm-intel.ko or kvm-amd.ko*. KVM also requires a modified QEMU although work is underway to get the required changes upstream.

	XEN-para VM	XEN-hvm VM	KVM VM
Processor	1 vcpu	1 vcpu	1 vcpu
RAM	2 GB ram	2 GB ram	2 GB ram
HD	disk on a file	disk on a file	disk on a file
Net		“netfront” network driver	e1000 network driver emulation
HV Version	XEN v.3.2.1	XEN v.3.2.1	KVM v.83

Table 2.3: Virtual machines specifications.

There are several studies that show comparisons of the hypervisors [31], in the Figure 2.1 is shown the comparison of XEN, KVM and the host with the measure of CPU and the HEP-Spec06 [78] benchmarks, which is the standard in Hight Energy Physics (HEP) community for CPU performance benckmarking and it is a based on subset of *Spec benchmark* [116]. The Figure shows an increasing number of virtual machines that are running concurrently on the host, and the results confirm that we can “overload” the cores of our machine and run 8 VMs concurrently, without any significant performance loss. Table 2.3 shows the VMs specs in the benchmarks. The OS installed are for the Host, "OS Scientific Linux 5.2, kernel 2.6.18-92.1.22.el5" and for the VMs "OS Scientific Linux 4.5, kernel 2.6.9-67.0.15.EL.cern".

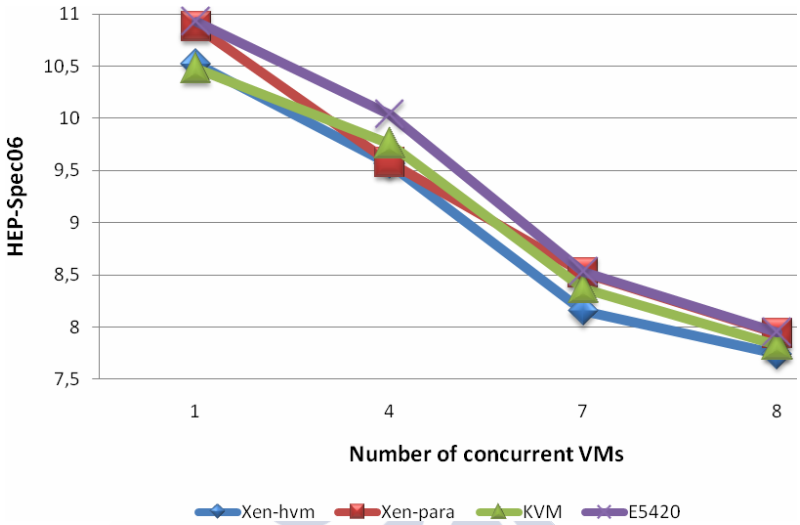


Figure 2.1: Hipervisors comparison with HEP-Spec06 Benchmarks.

Table 2.4 shows the exact percentage loss compared to physical CPUs. With virtualization technologies, either XEN or KVM, the loss is comparable to a CPU downgrade (to the preceding model). The benchmark specifications were running in an dual *intel E5420* processor, 16GB ram and two 10k SAS disks connected to a LSI Logic RAID controlled.

Virtualization Technology	**% loss from non Emulated CPU (E5420, 8VM)**
E5420 KVM	3.42
E5420 XEN-HVM	4.55
E5420 XEN-Para	2.02
E5410 vs. E5420	4.07

Table 2.4: VM lost of perfomance.

2.1.2 Cloud managers

Cloud computing is a service that relies on a highly virtualized physical infrastructure. In the Cloud, applications generally run on virtual servers that are independent of the underlying hardware. But there's more to the Cloud than virtualization, in that Cloud Computing is based on the concept of a *"utility computing"* service, where RAM, CPU cycles, storage and

network bandwidth are commodities to be consumed on a *"pay per use"* basis, like water or electricity.

A Cloud Computing environment relies on many physical and virtual servers. It is configured both in hardware and software to provide high reliability and availability. Clouds are also very flexible and scalable, in the sense that an application simply consumes resources as needed. Two such public Clouds computing services are *Amazon's Elastic Compute Cloud (EC2)* service and *Microsoft Windows Azure Platform*; in addition, similar services are offered by a number of web hosting providers, referred to as *Virtual Private Servers (VPS)*. In each of these services customers are charged by the amount of time their virtual machine is running (in hours or months), rather than by the amount of CPU time used. The benefit of Cloud Computing is that the provider takes care of the infrastructure your application runs on. You can eliminate the start up costs associated with hardware, and the infrastructure of a cluster is able to support multiple platforms dynamically.

Currently, there are several Cloud manager **open-source solutions**, which you can chose depending on you want to do. Some of the main important are *OpenNebula, Eucalyptus and CloudStack*.

The OpenNebula Cloud manager allows building any type of Cloud: public, private, and hybrid. It was designed to be integrated with any type of network and storage, and can be adapted into any existing datacenter. OpenNebula can be managed through a Web management server that provides access to all features and it has a powerful **REST-Full-API** that allows accessing all features too. It adopts classical cluster-like architecture with a front-end and a set of hypervisor-enabled cluster nodes where VMs will be executed. The main components of an OpenNebula Cloud are: the front-end, nodes, the image repository, the OpenNebula daemon, and the drivers.

OpenNebula supports two user roles, privileged user that can manage the OpenNebula Cloud and not privileged users that can manage their own virtual networks and VMs.

OpenNebula support the hypervisors XEN, KVM and VMWare ESX [133]. It employs a *Virtual Machine Definition File* (VMDF) to launch a VM. The VMDF is a template defining a set of attributes to specify a VM, and it can be managed through the *Command Line Interface* (CLI) whose syntax is similar to the Linux shell.

OpenNebula supports several Cloud client interfaces such as Amazon EC2, OGF OCCI and vCloud interfaces. The most important feature of OpenNebula is the contextualization

which allows modifying on boot the configuration parameters of the VM, such as, DNS, hostname, or even adding user SSH keys.

The Eucalyptus Cloud manager allows building private and hybrid Clouds adding external Cloud resources. Eucalyptus has five types of components: the Cloud Controller, Walrus, the Cluster Controller, the Storage Controller, and the Node Controller. As OpenNebula it manages 2 roles of users, one with privileges to manage the Cloud and the other without privileges to manage their own virtual networks and VMs. The hypervisors supported are KVM and XEN. It has specific images called Eucalyptus Machine Images (EMIs) that are a combination of a disk image, a kernel, a ramdisk, and a XML file containing metadata about the image. It can be managed through the eucatools CLI whose syntax is similar to the Linux shell. Eucalyptus provides a client interface that is compatible with Amazon EC2 and S3 [3] Web Services.

Finally, CloudStack is an open-source software architecture, which permits building any type of Cloud: public, private, and hybrid. It allows the deployment and management of elastic computing environments. It has four types of components, *the Management Server, Pods, Availability Zones, and Compute Nodes*. As a difference to the others, it has 3 user roles; *root administrator, domain administrator, and not privileged users*. The domain administrator new user has the peculiarity of performing administrative operations for users who belong to that domain. The hypervisors supported are XEN and KVM. It uses *Qemu Copy-On-Write QCOW* [119] format as virtual machine disk images. As OpenNebula, CloudStack can be managed through the **web management server** and it has a **powerfull REST-Full API** [33] to access all features of the Cloud manager.

2.1.3 Cloud managers study

The first step towards providing a distributed infrastructure solution to support multi-site environments for scientific research is to analyse the features of the available open-source [69] Cloud managers. *OpenNebula 2.0Beta, Eucalyptus 2.0.1 and CloudStack 2.1.4* were the software versions which were analysed. Fig 2.2 show the features of the analysed Clouds.

The categories have been grouped in two subsets, the first one contains the basic features that must be present in all Cloud management platforms and the second one defines the advanced characteristics that are desirable in Cloud managers. And as final results of this analysis CloudStack is the one with the highest score in terms of features, and it is followed by OpenNebula. About CloudStack it is remarkable the web interface that provides the complete

Requirement	Description	OpenNebula	Eucalyptus	CloudStack
Basic features				
User management	Create, delete, show, authenticate, password management	✔	✔	✔
User roles	Administrator	✔	✔	✔
	Domain administrator			✔
	Unprivileged	✔	✔	✔
Node management	Create, delete, show	✔	✔	✔
	Activate, deactivate	✔		✔
	Linux node support	✔	✔	✔
	Windows node support			
Template management	Register, unregister, modify, show, publish templates	✔	✔	✔
	Activate, deactivate	✔	✔	
	Predefined templates		✔	✔
Virtual network management	Create, delete	✔	✔	✔
	Show	✔		
	Port forwarding, load balancing			✔
VM management	Start, stop, create, delete	✔	✔	✔
	Show graphic guest VM environment			✔
	Quick instantiation		✔	
	Migrate	✔		✔
	Suspend, restore	✔		
	Script execution on boot	✔		
	High availability VM			✔
	Install VM from ISO image			✔
Hypervisor support	Xen support	✔	✔	
	Xen Server support			✔
	VMware Player support			
	VMware ESXi support	✔		
	VirtualBox Support			
	KVM support	✔	✔	✔
Advanced features				
Virtual volume management	Create, delete, connect and disconnect		✔	✔
ISO image management	Create, delete, connect, disconnect, copy			✔
Guest contextualization	Linux guest contextualization	✔		
	Windows guest contextualization			
Cloud deployment model	Private cloud	✔	✔	✔
	Public cloud	✔		✔
	Hybrid cloud	✔	✔	✔
Cloud interfaces	Amazon S3 support		✔	
	ElasticHost support	✔		
	EC2 Query interface	✔	✔	
	OGF OCCI interface	✔		
	vCloud interface	✔		
Storage management	Add, remove storage			✔
Domain management	Define, edit, delete, and set quota limits			✔
Monitoring management	Network monitoring			✔
	Node monitoring	✔		✔
	VM monitoring	✔	✔	✔
	Service monitoring			
	Storage monitoring			✔
Administrative alert registration	Register, show administrative alerts			✔
Event registration	Event registration			✔

Requirement	Description	OpenNebula	Eucalyptus	CloudStack
Common requirements				
User management	Create, delete, show, password management	✔	✔	✔
Node management	Create, delete, show nodes	✔	✔	✔
Template management	Register, unregister, modify templates	✔	✔	✔
VM management	Start, stop, delete, virtual machines	✔	✔	✔
Hypervisor support	Xen and KVM support	✔	✔	✔
Requirements for a supercomputing center (private cloud)				
Multiple Hypervisor support at same time	Support for multiple hypervisors at the same time	✔		
Contextualization on boot	Modify the configuration parameters of the VM on boot	✔		
Script execution on boot	Execute scripts automatically in VM guest at booting	✔		
Monitoring	Monitor nodes and VMs	✔		✔
Requirements for a company (hybrid cloud)				
Amazon EC2 support	Use Amazon EC2 resources		✔	
Amazon S3 support	Use Amazon S3 resources		✔	
Requirements for an organization (public cloud)				
Install VMs from ISO image	Install VMs using ISO images			✔
Access to VM graphical interface	View the VM graphical interface without installing additional software			✔
Access to cloud features by Web	Manage cloud using a Web interface			✔
		36	27	40

Figure 2.2: Cloud managers features.

management of the Cloud, both for the system administrator and for the unprivileged user, at the same time it provides useful information such as monitoring data, resource utilization statistics, registering information and alerts. Other important point for the infrastructure is the way it defines the availability of VMs that the system maintains operational without user or administrator intervention at all. Furthermore, it allows creating a Cloud with a simple interface to offer it to internal or external users.

2.2 Big Data Overview

In [37], Google introduced MapReduce programming model offering an efficient way of performing distributed computation over large data sets. After that, Big Data technologies has become the most interesting option for data-intensive analyses in the cloud and in commodity clusters due to its excellent fault tolerance features, scalability and the ease of use. Although it have been argued that MapReduce does not suit well for many scientific algorithms, several works [44], [72] have shown that the MapReduce model can be used successfully even for solving complex scientific computing problems. Different implementations of the Google's original MapReduce model [37] has been proposed, like Twister [43] for iterative computing or Phoenix++ [118] for shared-memory systems, being Hadoop [5] and Spark [143] the most popular ones. Several extensions have been done to use MapReduce in different architectures like GPUs [48], [80], manycores [93] or HPC [47], [94].

In the cloud, AWS EMR [10] is the best known example of Hadoop-as-a-Service. Other cloud based implementations are Azure MapReduce [72], qosh [113], Sahara [112], Dynamic MapReduce [92], or Cloud MapReduce [91].

There are several projects that have addressed the Hadoop federations, allowing the MapReduce tasks to run on multiple clusters. In [134], the Hadoop MapReduce framework was extended for this purpose, reusing the user authentication and job submission mechanism of Hadoop. Hadoop has also created the Hadoop File System (HDFS) Federation [6], for scaling horizontally, and the federation of multiple and independent NameNodes. Other works like [85], have created with a front-end the Hadoop data management over several clouds, and new Hadoop architectures like [111] for federation have tried to emulate to Amazon EC2, provisioning Hadoop clusters and submitting jobs, allowing also users to focus on writing their MapReduce applications rather than managing cloud resources.

In addition to extending DIRAC for Big Data technologies, we propose global scheduling by load balancing (LB). The concept is sensible, following other LB approaches in similar distributed computing environments. LB between different cloud providers has been demonstrate to save costs [122]. Both job completion and LB are targets of max-min strategy for job scheduling in cloud computing environments [45], including QoS constrains. Furthermore, there are a variety of algorithms for distributed computing LB, like the Hierarchical Load Balanced Algorithm (HLBA) for Grid environment [88] or the Dynamic Load Balancing Algorithm (DLBA) [117] or DLBA with genetic algorithms in [89]. These works motivate the use of LB as a sensible approach to deal with the global scheduling in DIRAC as a Big Data federation.

2.3 DIRAC Evolution

The **Large Hadron Collider** (LHC) is the world's largest and highest-energy particle accelerator. It was built by the **Centre of European Organization for Nuclear Research** (CERN). Four big detectors have been constructed at the LHC, and the **Worldwide LHC Computing Grid** (WLCG) project was created with the mission of analyse all raw data generated by the LHC. Furthermore, most of the computing resources needed by the LHC experiments as well as for some other communities are provided by Computing Grids. Grids provide a uniform access to the computing and storage resources which simplifies their usage. The Grid middleware stack offers also the means to manage the workload and data for the users. However, the variety of requirements of different Grid User Communities is very large and it is difficult to meet everybody's needs with just one set of the middleware components.

For LHCb, DIRAC was developed as a Grid middleware [40]. It was designed to be a generic system with LHCb specific features well isolated as plug-in modules. It allows to construct medium sized Grids of up to several tens of thousands processors by uniting PC farms with most widely used cluster software systems as well as individual PCs within its integrated Workload Management System. Next sections provides an brief overview of Grid Computing DIRAC and the evolution from middleware to *interware*.

2.3.1 Grid computing evolution

The Grid Computing is the federation of computer resources from multiple administrative domains to reach a common goal. *Ian Foster*, an expert in the field defined the Grid Computing

as a *"hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities"*. He later clarified this definition by creating a three-point checklist [59] that characterizes a Grid as a system that, first *coordinates resources that are not subject to centralized control*, second *uses standard, open, general purpose protocols and interfaces*, and finally *delivers non-trivial qualities of service*.

Another similar vision of Grid Computing was made by Baker [11] that present four basic characteristics that define a Grid:

1. **Multiple administrative domains and autonomy**. Grid resources are geographically distributed across multiple administrative domains owned by different organizations. The autonomy of resource owners needs to be honoured along with their local resource management and usage policies.

2. **Heterogeneity**. A Grid involves a multiplicity of resources that are heterogeneous in nature and will encompass a vast range of technologies.

3. **Scalability**. A Grid might grow from a few integrated resources to millions. This raises the problem of potential performance degradation as the size of Grid increases. Consequently, applications that require a large number of geographically located resources must be designed to be latency and bandwidth tolerant.

4. **Dynamicity or adaptability**. In a Grid, resource failure is the rule rather than the exception. In fact, with so many resources, the probability of some resource failing is high. Resource managers or applications must tailor their behaviour dynamically and use the available resources and services efficiently and effectively.

This Grid vision requires protocols, interfaces, and policies that are open, general purpose, and standard. The standard allows users to establish resource-sharing arrangements dynamically with any interested party to create something more *"that a plethora of balkanized, incompatible, non-interoperable distributed systems."* The commercialized embodiment of this work has been the **Globus Toolkit**.

Looking back at the evolution of Grid technologies, one has to mention an important change that followed the proposal of the **Open Grid Service Architecture** (OGSA). The OGSA was introduced in the paper *"The physiology of the Grid"* by I. Foster , C. Kesselman, J.M. Nick and S. Tuecke in 2002 [60]. OGSA introduced the notion of a Grid Service, that

is, an extended Web Service [135] with a more specific semantics tailored towards handling state, persistence and life cycle management.

Besides OGSA, over the last decade, a vast number of architectures, protocols and standards were created by the Grid community and, consequently today Grid technologies form a whole stack of various components. To facilitate the development of Grid-related standards and tools, the community formed the **Global Grid Forum** (GGF). In 2006 the GGF merged with the Enterprise Grid Alliance forming the **Open Grid Forum** (OGF) which continues to be an important forge of Grid standards and common APIs. One of the most notable recommended standards prepared by the OGF is the **Distributed Resource Management Application API** (DRMAA) [101]. The adoption of DRMAA allows for interoperability between different Grid frameworks.

Figure 2.3 shows Globus Toolkit components with several Grid frameworks. Globus Toolkit was the first open source framework that enabled to create Grids. The first stable release (1.0) appeared in 1998. According to *Tannenbaum* the toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications [1]. Globus focuses on interoperability and thus defines a set of protocols and common components that form the Open Grid Service Architecture (OGSA). The details describing the basics of the toolkit were provided by Foster et al. in papers entitled *"The Anatomy of Grid"* [61] and *"The Physiology of the Grid"* [60].

The Globus Toolkit was developed by companies, organizations and individuals gathered around the Globus Alliance that was officially created in 2003 to formalize and bring together developers formerly working on the Globus project.

Another important framework was **UNICORE**, which stands for **UNiform Interface to COmputing REsources**. It is an open source Grid framework developed and used primarily in Europe. The project was initially funded by the **German Ministry of Education and Research** (BMBF) and mainly evolved around the German Supercomputing Centre, Forschungszentrum Jülich. The initiative was started in 1997 as an attempt to integrate the infrastructure of German supercomputing centres. The main characteristics of UNICORE project [127]:

1. Open source under BSD license.

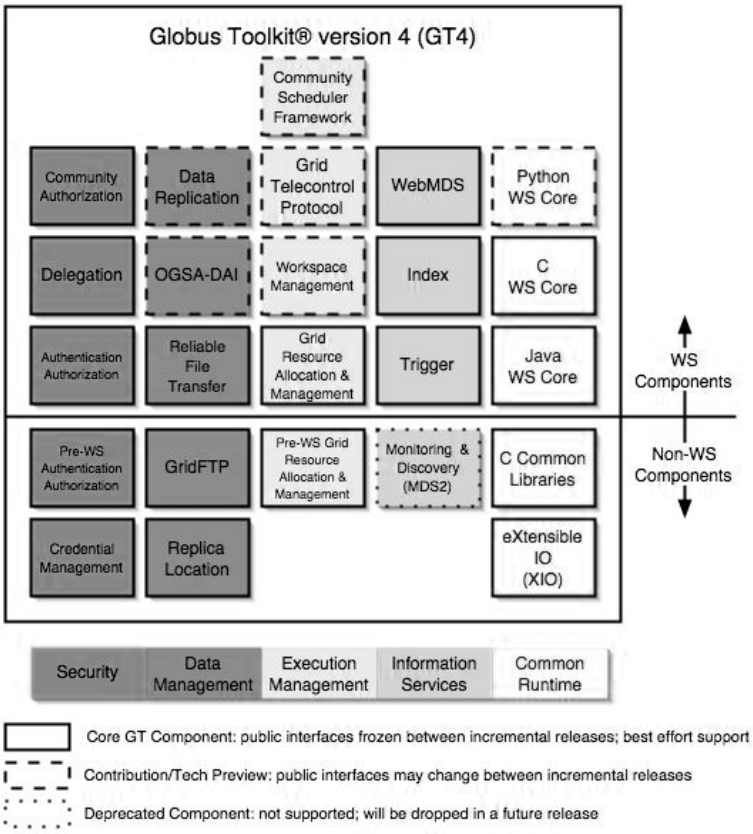


Figure 2.3: Globus Toolkit components.

2. Standards-based, conforming to the latest standards from the Open Grid Forum (OGF), W3C, OASIS, and IETF, in particular the Open Grid Services Architecture (OGSA) and the Web Services Resource Framework (WS-RF 1.2).

3. Open and extensible realized with a modern Service- Oriented Architecture (SOA), which allows to easily replace particular components with others.

4. Interoperable with other Grid technologies to enable a coupling of Grid infrastructures or the users needs

5. Seamless, secure, and intuitive following a vertical, end-to-end approach and offering components at all levels of a modern Grid architecture from intuitive user interfaces down to the resource level. Like previous versions UNICORE 6 seamlessly integrates in existing environments.

6. Mature security mechanisms adequate for the use in supercomputing environments and Grid infrastructures. X.509 certificates form the basis for authentication and authorization, enhanced with a support for proxy certificates and virtual organizations (VO) based access control.

7. Workflow support tightly integrated into the stack while being extensible in order to use different workflow languages and engines for domain-specific usage.

8. Application integration mechanisms on the client, services and resource level for a tight integration of various types of applications from the scientific and industrial domain.

9. Different clients serving the needs of various scientific communities, e.g. graphical clients to define complex workflows, command line tool, web-based access.

10. Quick and simple to install and configure to address requirements from operational teams and to lower the barrier of adopting Grid technologies. Similar the configuration of the various services and components is easy to handle.

11. Various operating and batch systems are supported on all layers, i.e. clients, services and systems; Windows, MacOS, Linux, and Unix systems as well as different batch systems are supported such as LoadLeveler, Torque, SLURM, LSF, OpenCCS, etc.

12. Implemented in Java to achieve platform independence

Glite [66] was implemented as part of the **EU-sponsored Enabling Grids for E-Science in Europe** (EGEE) project that started in 2004. After prototyping phases in 2004 and 2005, convergence with the LHC Computing Grid (LCG-2) distribution was reached in May 2006 when gLite 3.0 was released. It was born from the collaborative efforts of more than 80 people in 12 different academic and industrial research centres as part of the EGEE Project. GLite provides a framework for building Grid applications tapping into the power of distributed computing and storage resources across the Internet. The gLite is a service-oriented Grid framework, it is a middleware platform designed from the beginning to support the Grid built

within and around CERN. The gLite Grid services follow a Service Oriented Architecture which facilitates interoperability among Grid services and allows easier compliance with upcoming standards, such as OGSA, that are also based on these principles. The services are expected to work together in a concerted way in order to achieve the goals of the end-user, however, they can also be deployed and used independently, allowing their exploitation in different contexts.

The gLite project ended in 2010, the development of the gLite middleware was then taken over by the European Middleware Initiative, and is now maintained as part of the **European Middleware Initiative** (EMI) [46] software stack. The distributed computing infrastructure built by EGEE is now supported by the **European Grid Infrastructure** (EGI) and runs the Grid middleware produced by the European Middleware Initiative, many components of which came from the gLite middleware, UNICORE [127] or dCache [36].

In order to summarize this section, the examples given show that Grid Computing is a relatively widely adopted and well established technology that introduced a significant number of architectures, standards and protocols. However, despite the immense diversity among different Grid technologies, there are some general shortcomings which are still present nowadays. One of the most important deficiencies is the complexity of the available architectures and solutions that make them difficult to deploy and manage. Another important aspect is the lack of stability that may result from the fact that Grids emerged as a technology made for scientists by scientists and despite being around for more than a decade they still lie primarily within the domain of science. Another important issue with current Grid technologies involves the high complexity, instability, potential single point-of-failure due to static configuration and impaired performance as a result of using XML and SOAP. Section below describes the federated metacomputing approach followed in this research and shows how these issues are addressed in the architecture of a **Federated Clouds Environment**.

2.3.2 DIRAC Interware

The **DIRAC project** was initially designed, in 2003, as a community management software to support scientific computing in the *LHC context* [124]. DIRAC was created as a Grid middleware to support the Grid community of the LHCb experiment and at the beginning with the specific use case of Monte Carlo (MC) simulation. Although developed for the LHCb experiment, it is designed to be a generic system with LHCb specific features well isolated as plug-in modules. The DIRAC architecture consists of numerous cooperating *distributed*

services and *light agents* built within the same **DIRAC SEcure Transport** (DISET) [26] framework following the Grid security standards. DIRAC introduced the now widely used concept of **Pilot Agents** [27]. This allows efficient **Workload Management Systems** (WMS) to be built. The workload of the community is optimized in the central Task Queues. The WMS is carefully designed to be resilient to failures in the ever changing Grid environment.

The DIRAC Project has all the necessary components to build Workload and Data management systems of varying complexity. It offers a complete and powerful Grid solution for other user Grid communities and due to this characteristics DIRAC is the perfect tool to integrate with the infrastructure of this thesis.

Nowadays, DIRAC is used by LHCb as the management middleware of the community, including not only MC, but all the other computing activities such as the reconstruction of the experiment data collection, the selection of events by different filters, the reprocessing of that data, the user analysis or the data transfers. Moreover, other scientific communities besides LHCb are using DIRAC such as ILC, Belle II and others [123]. Some of them have extended the DIRAC core functionality to fulfil their requirements [71]. These experiences have been useful to validate two important characteristics of the software: the robustness of the framework, and an extensible design, resulting in a solid software stack able to deal with different use cases in a flexible manner.

DIRAC offers different tools to provide an integral solution for workload and data management on distributed computing infrastructures. From the beginning of the project, the need to provide a user friendly interface in the context of Virtual Organizations (VOs) was clearly identified. To fulfil this need, DIRAC provides a *web portal* offering a friendly *multi-user and multi-VO GUI*, matching the different users operations, rights and permissions, including the computing administration in different layers, the final user analysis, or the anonymous view of the public domain. The DIRAC portal also offers the possibility to integrate other web portals.

Using DIRAC one can deal with the overlaying middleware in a transparent way. DIRAC was designed to make use of Grid and non-Grid technologies. For example, at the beginning of the project in 2003, the distributed resources were only available to LHCb via direct submissions to the **Local Resource Manager System**s (LRMS). Later, the Grid technologies evolved towards offering Grid interfaces for remote access to batch queues. For these reasons DIRAC can be defined as an **interware** that allows interoperability among different computing resources via different drivers to each of the middleware solutions. Some of the

new drivers developed are in the field of Cloud Computing in order to extend the *DIRAC interoperability*, starting from the Belle experiment using *Amazon Elastic Compute Cloud* (EC2) [71].

The following are some of the DIRAC main components:

1. Secure Transport: DIRAC components use a custom protocol for client-server communication; the **DIRAC SEcure Transport** [26]. This protocol provides a uniform way for building distributed applications which are connected securely through X509 certificates [58] and Grid proxies [108] for authentication. DIRAC components on the Cloud will also use the DISET protocol for communicating with other parts of the system to ensure the security of the setup.

2. Configuration: The **Configuration System** (CS) is the set of DIRAC components that provides configuration of the rest of the DIRAC framework. CS includes the configuration system itself, to locate each other using **Service URLs** (SURLs). At the same time, it allows global default values for configuration parameters of any DIRAC component to be set. DIRAC configuration options are organized in a tree structure where each node in the tree is a section that contains options and/or other sections. Each option gets an associated value that can be retrieved with a single call inside the DIRAC python code.

3. Workload Management: **The Workload Management System** (WMS) is one of the central pieces of the DIRAC framework. It is based on a pull scheduling paradigm with late binding of resources to payload. DIRAC implements the pull scheduling as follows:

   – A central WMS server keeps the list of pending tasks organized in TaskQueues.

   – Each TaskQueue contains tasks with identical execution requirements, e.g., CPU time requested by the user, the required platforms to execute, or the identity of the submitter.

   – Dedicated DIRAC agents attempt to get hold of appropriate computing resources for each TaskQueue. This is done via pilot job, see jobs [27], both when using Grid resources, local batch systems or Clouds.

   – Once a pilot manages to execute at a certain resource, a WN, it checks the environment and requests a pending task from the central server.

- After the execution of the task is completed, the pilot uploads the results, reevaluates the computing resources available and requests a new pending task if appropriate.

4. **Data Management**: There is a limited set of components from the DIRAC **Data Management System** (DMS) that are relevant for the proposed setup. They are the following:
   - **Storage Element** (SE): DIRAC provides a secure remote interface to the local storage of a server via the Storage Element service using the DISET. It is mostly used for hosting input/output sandboxes for WMS tasks.
   - **File Catalog** (FC): DIRAC provides a **Replica File Catalog** server implementation including access control and trivial metadata like file size, checksum or creation timestamp. Additionally DFC provides a complete metadata catalog solution fully integrated with the replica catalog. DIRAC also provides abstract interfaces to third party SE and FC solutions for early integration in the final system.
5. **Web Portal and Other Components** With the **Web Portal System** DIRAC provides the possibility to monitor and control the activity of the system and its behaviour via a dedicated web portal. Connections to the portal are encrypted and the user is authenticated by the user's Grid certificate that it is imported to the browser.
6. **Accounting** Provides a persistent backend and report generation tool for the information relative to the different actions controlled by DIRAC such as user task execution or data upload and registration.
7. **User Interface** Allows the authorized user to submit the Tasks and retrieve their outputs.

The previous list shows that DIRAC is a suitable software with the characteristics needed to create a scalable system to submit research jobs and that can easily integrate Cloud resources. The other part now is the software distribution, when the job is submitted to some SITE or computer lab in order to be executed without penalties problems.

2.4 Software Distribution Systems

One important part in distributed computing is the access to the software that is needed for execution of the researcher's applications. Nowadays EGI has established a large production Grid with thousands of users belonging to different communities that are organized in hundreds of **Virtual Organizations** (VO). Each of these VOs requires different applications that ideally should be available in the nodes where they are going to run. Research groups processing data in distributed systems often need specific software. This software needs to be downloaded and installed in the local operating system in order to correctly execute the user jobs. It is therefore interesting to develop an infrastructure that allows to automatically download the software, with the possibility of being operated externally by a self-updating version manager, which could be used with virtualized environments. Different SITES have different alternatives to distribute the software of each VO. The most common way is to use a NFS directory shared between all the nodes in the cluster. This approach has several drawbacks especially for large clusters. First, it imposes an important performance penalty when all the nodes try to access the same NFS server. In addition, it seriously limits the scalability of the system.

One option is to use several NFS servers, splitting the software repositories among them. But this solution is more difficult to deploy and requires some planning to decide how the different software repositories will be splitted between the different servers. Another option is to use NFS caching abilities that appear in the last version of **Red Hat Enterprise Linux** based systems (RHEL6), but most SITES are still using operating systems based on RHEL5.

There is also the option of using a distributed file system like **OpenAFS** [102], but it requires the kernel installed in the nodes to support it, and it usually involves a complex configuration process.

In addition, the software area management in each of the SITES is a redundant administration task, introducing inconsistencies between the publication in the software area of the different SITES. Ideally, one aims to have a single administration of the software area for all the SITES.

Another option is the use of **TARBALL files**, which are encapsulated software files compressed in **Tar** [120] and **Gzip format** [74]. The distribution of theses files is done through *HTTP servers*, these files are downloaded to the worker nodes and they are decompressed to run the scientific software. But depending on the number of Nodes, the download latency could be a serious problem in the execution time of the scientific software.

There are programs that solve some these problems, and can help in these tasks as **Cern Virtual Machine File System** [30] (hereinafter CernVM-FS), which provides the necessary software without concern for the user. CernVM-FS is a software designed to easily recover files from an HTTP server. This solution uses file catalogs to establish read-only file systems through HTTP protocol. Files are first transferred to the local Squid proxy, if available, and then to the local cache of the node; greatly reducing the network traffic in case of jobs that use the same software. Software distribution will be explained in more detail in chapter 4.

2.5 Virtualization and HEP Software

Cloud promised to address with the same shared set of physical resources a large user base with different needs. However, the virtualization could be an issue if the performance of the applications is being decreased excessively, and an important question arises: **is the virtualized performance of Clouds sufficient for scientific software?** Albeit early attempts to characterize Clouds and other virtualized services [90] [12], this question remains largely unexplored. One of the chapters of this thesis will try to analyse systematically the performances in virtualized platforms, of different setups and several types of software used in the HEP field.

The majority of Cloud projects are using for virtualization one of the two main open-source hypervisors **Kernel based Virtual Machine** [90] and XEN [139]. In addition, most of virtualized SITES are using KVM with their Cloud managers configuration. In the case of KVM, it is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (**Intel VT or AMD-V**). It consists of a loadable kernel module, *kvm.ko*, that provides the core virtualization infrastructure and a processor specific module, *kvm-intel.ko or kvm-amd.ko*. KVM also requires a modified *QEMU* although work is underway to get the required changes upstream [61]. There have been studies that characterized the effectiveness of Cloud technologies and virtualization over competing technologies. In this vein L. Youseff et al. [142] did a study of the performance of paravirtualized systems based on XEN with benchmarks from HPC Challenge, LLNL ASCI and NAS parallel suits. An application for simulation of oceanographic and climatological phenomena was also used as a benchmark. For most of the tests they found no overhead when using a virtualization environment. Meanwhile, E. Deelman et al. [38] did an study of the costs of using a commercial Cloud services versus the cost of having a dedicated infrastructure. They estimated the cost by running an

astrophysics application to compute mosaics of input images, obtaining the amount of CPU used by the runs, the size of the input and output files and multiplying these numbers by the costs of a CPU-hour, GB of storage, input transfer and output transfer. Their results show that commercial Clouds could be an interesting alternative for certain class of scientific jobs. Finally, A. Chierci et al. [31] did a qualitative study performance of XEN versus KVM by changing a computing element and a monitoring from XEN, the technology used for all the machines LHCb TIER-2 they are hosting at CNAF, by two KVM machines. Although the users were not informed, they did not notice the change in the more than 3 weeks that the test was run. A second part of their study consisted on comparing two XEN, paravirtualized and a hardware VM, and one KVM virtual machines. The **HEP-Spec06** [78] test, based on a subset of the *SPEC benchmark* is a standard benchmark suite in HEP community used to measure CPU performance, *iperf* and *bonnie++ tests* were used to do the quantitative comparisons. The intent of this thesis is to obtain a quantitative evaluation of the overhead of using a KVM VM to run HEP software applications instead of a bare metal machine which has not been addressed by these previous works. This evaluation of the performance is an important issue to answer the question of wether the Cloud is a sufficient performance solution for Scientific software.

2.5.1 HEP software description

Three computer programs have been selected to make the test: (i) **GAUSS** which is a specific software developed for the LHCb experiment; (ii) **AIRES** developed for Auger, an experiment to study cosmic rays; and (iii) a program developed by one of the groups at the Instituto Galego de Física de Altas Enerxías (IGFAE) [8, 7], which are belonging to ALICE experiment and makes use of FastJet. **FastJet** is software that contains jet finding algorithm used by major experiments like ATLAS, CMS, DØ, CDF, and others. In the following section more details of these computer programs will be given.

The software used in HEP experiments can be classified in three categories: *simulation, event reconstruction and physic analysis*. Simulation software is used in all stages of an experiment, from the design to the commissioned phases. In the design phase, it is used to obtain some estimates needed for making detector choices, also in the development of the reconstruction and analysis programs and, finally, helps to evaluate the physics potential of the detector. Once the detector is commissioned, these programs are used to look for unexpected effects and backgrounds or, when a theory is not well known, to compare the results with

various models. Simulations are also used to make studies of the detector performance and to provide efficiency values for the data analysis. The second group is the software that reconstructs the trajectories followed by the different particles produced in a collision. This is not a trivial task because, even in the case that there were just one initial collision, the high energies involved imply a high numbers of particles and, therefore, a high number of simultaneous active sensors that have to be matched to the different particle tracks. The third group is the software that performs the physic analysis, this is the most heterogeneous group of the three since apart from the software used to select the interesting events for a given study it includes customized software developed to do the actual physic analysis of the reactions under study. The tests ran for this thesis belong to the first and third categories. Thereby *GAUSS and AIRES* belong to the simulation class software whereas *FastJet* software belongs to the physic analysis category. The AIRES package [115] was developed for the Auger experiment with the objective of simulating the production of particle showers created by the collision of ultra high energy particles present in cosmic rays, some of them with energies well over a million times the highest energies attainable at LHC, with the Earth's atmosphere. Because this code generates most of the needed data via algorithms, the IO load is low compared with the CPU load. This IO load is related to the input of the simulation parameters, the output of the results, logs and some summary files, and the update of the **internal dump file** (IDF), which contains the necessary internal data to restart a broken simulation from the last update of this file.

The second program used for the tests is GAUSS [42] that is a Montecarlo simulation program developed by the LHCb collaboration. The program uses the GAUDI framework, also developed by LHCb, that simplifies its interface with other programs. GAUSS is used to generate the collision that is going to be studied. For this purpose, it can make use of programs like Pythia [110], EvtGen [83], etc. This simulation is done by GEANT4 [34] a program developed for the simulation of the passage of particles through matter and which is currently used in HEP experiments, astroparticle physics, nuclear physics, space and medical applications, to name a few fields. These simulations are the most CPU consuming events in HEP experiments. For example, a typical GAUSS simulation takes 50000 SPECInt2000 sec/event while reconstruction takes of the order of 20 times less [64]. Just as with AIRES, the IO load of GAUSS comes, mainly, from the reading of the configuration files, and the output of the results, logs and summary files. Unlike AIRES, that does not need to be supplied with information about the distribution of matter in the volume of simulation because it uses

a model of the atmosphere that generates this information, GAUSS needs to read this information because the complex distribution of matter inside the detector can not be modelled by a simple algorithm. Nevertheless, this information needs to be read just once, independently of the number of collisions that a given GAUSS run will simulate. Another difference with AIRES is that GAUSS does not need checkpoints because it only takes about a couple of minutes to generate a single event.

The last program used corresponds to the analysis category. It was developed by the phenomenology group of the IGFAE to study the contamination of jets by background particles in heavy ion collisions. Jets are groups of particles, produced in the disintegration of high-energy particles, whose trajectories lie within an small angle cone. In [7], two programs were considered, but we have just used the one based in the FastJet [23] package. This package, via a plug-in architecture, has access to a plettora of algorithms ranging from those taking $O(N^3)$ steps, where N is the number of particles to analyse, to those taking just $O(N \log N)$. In this case the IO load is also small compared to the CPU load of the job despite the fact that the amount of input data that has to be read is considerably higher than in the previous cases.

As part of this thesis an evaluation of the usefulness of the current Cloud Computing services for scientific applications will be presented. The **Kernel-based Virtual Machine** (KVM) performance of a sample of HEP software under different setups is analysed with the use of a **multivariate analysis of variance** (MANOVA).

2.6 Federated Metacomputing

Catlett and Smarr have defined the term metacomputing as *"the use of powerful computing resources transparently available to the user via a networked environment"* [28]. Their view is that a metacomputer is a networked virtual supercomputer. To a certain extent our usage of the term metacomputing still holds true to this definition apart from explicitly referring to *"powerful"* computing resources. Other terms have also been used to describe this computing paradigm, such as seamless, scalable, global computing, Grid Computing and more recently Cloud Computing.

The most significant conceptual difference between Grid Computing and metacomputing is the location of the main business logics of a program. While Grid Computing focuses on finding the best host to execute a program that contains the main logic, in metacomputing the main logic reside above in the metaprogram. The metaprogram is not executed by a

certain host but instead, by a whole network of services that federate dynamically and form the virtual metacomputer. In the case of Cloud Computing the logic is a mixture and extension of both concepts, because in the case of select the best host to execute a program, there are more parameter to consider such us the cost of the resources or the overload of the machine. Furthermore, in Cloud Computing it is needed to know the specific Cloud environment in order to create the appropriate instance.

	Grid Computing	**Federated Metacomputing**	**Cloud computing**
Architecture	Centralized/Client Server	Service-2-Service	Service-2-service with Cloud managers and client-server with WN
Task scheduling	Schedulers and queues	No schedulers or queues - resources are scheduling assigned dynamically at runtime from all available providers in the network	queues systems and dynamic resources creation
Managed by	Central resource manager/scheduler	Decentralized coordinating service/scheduler invoked per request (many instances may operate simultaneously)	Cloud managers and central resource managers
Configuration	Static using DNS or IP addresses	Dynamic using discovery/join protocols	Static using DNS or IP addresses
Remote invocation	RPC over web services/SOAP	Federated Method Invocation executing metaprograms	SOAP, RestFull and specific Cloud manager protocols

Table 2.5: Grid Computing, Metacomputing and Cloud Computing

Another important difference refers to the architecture, discovery of services and binding. While Grids are de facto built using **Client-Server Architecture**, the federated metacomputing is a true peer-to-peer service-to-service environment where all service providers are

located using dynamic zero-configuration discovery/join techniques, and in the case of the new Cloud Computing model, there are a mixture of services and resources where by one hand there is a client-server architecture in the manager of job submission side and the other side is the peer-to-peer service-to-service environment where the Cloud managers need to be known about other servers with specific protocols.

Detailed comparison of Grid, Cloud and Metacomputing is presented in Table 2.5.

The metacomputing environment offers functions at the level of the whole metacomputer in form of decentralized system services that handle security across all providers, distributed storage, distributed file system or services that coordinate the execution of complex metaprograms across the network. Taking into account the previous critical view of Grid technologies and this platform level approach, it should be noted that Grid technologies should be regarded as batch schedulers that move executable code around a network of statically connected computer nodes. They perform these high level system functions in a centralized manner. In contrast, in a metacomputing environment all resources are treated together in a dynamic, decentralized way, similar to an integration with a Cloud Computing model where for one side you should have a unique batch scheduler for job submission, but the resources have to be treated together in the way of to know the specific SITE characteristics.

2.7 Summary

The state-of-the-art and relevant literature that allows to set the approach taken and the contribution of this dissertation with respect to currently available technologies were presented in this chapter. As the introduction explained, the *Cloud Computing overview* in order to understand the main purpose of Cloud environments and some of the main characteristics of *open-source Cloud managers* to select the most suitable for our purpose in this thesis. The current ecosystem *Big Data software* environment was also presented. Furthermore, a brief introduction of the Grid Computing and some gaps were described, with some solutions based in Cloud Computing model. The *DIRAC Grid middleware*, its transformation to an *interware*, and the main components of this software were shown. Several distribution systems and the most suitable are for the purpose of this thesis were discussed. Several High Energy Physic software (HEP-Software), which KVM performance will be statistically analysed, were shown. And finally the details of *metacomputing* approaches, and how to the cur-

rent solutions do not address *federated metacomputing models* based on Cloud environments is described, which is one of the contributions in this thesis.

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

CHAPTER 3

SOFTWARE DISTRIBUTION AND VIRTUAL PERFORMANCE

This chapter contains the studies of scalable systems for the software distribution and the performance of HEP software in virtual environments.

3.1 Software distribution

This section contains the study to select an scalable system for the software distribution for the infrastructure of this thesis, describing the motivation and the problems of current software distribution, defining the deployment of the CernVM-FS repository with benchmarks [50] and finally showing how to create a Software Repository Service.

3.1.1 Motivation

Currently, there is a big number of researchers that need to run their jobs using distributed systems, and they need specific software, which is usually not installed in the worker node. The most of the times, the user is in charge of installing the software to run the simulations, and it increases the working time. This working time is the maximum time including the software installation in the node as shows the next equation:

$$\max_{i \in N_D} \left(\max_{j \in S_i} T_{ij} + t_i \right), \quad N_D = \{1, 2, \ldots, M\}, \quad S_i = \{1, 2, \ldots, N_i\}$$

$$N_i = \text{max jobs in i node}$$

$$T_{ij} = \text{total time of software execution}$$

$$t_i = \text{software installation time}$$

$$i = \text{node of running jobs}$$

$$j = \text{number of jobs per node}$$

As it has been shown, with the increasing the time due to software installation, it will increase the amount of total finish time of the user jobs. The computing in distribution systems needs reliable and efficient systems to get the software that the researchers need to run their programs. The Worldwide LHC Computing Grid [137] provides a distributed infrastructure for processing the data of the Large Hadron Collider [121]. The model currently in use consists in executing some special jobs at each SITE that install locally the software in a shared area. Such jobs are executed under particular conditions in order to give them high priority, as they are essential for the usability of the SITE and they must be assigned to a particular WN of the farm for which the shared area is mounted in write mode. The WNs can then access it through the local network through NFS protocol. Problems related to this procedure for software distribution happen very often and make the SITE unavailable for running jobs. The most frequently observed issues concern the shared area availability and the software installation jobs.

The shared area is made available to the WNs through NFS (or AFS) protocol. This type of setup presents several limitations: any outage of the NFS/AFS server makes the content of the shared area inaccessible and immediately causes the failure of all running jobs. Also, in case of high load, the NFS/AFS server can become very slow, causing the jobs to time out in the execution of the application and eventually fail. In a infrastructure where computer classrooms are included, it would be necessary one NFS per class in order to make an scalable solution. Even if, in principle, SITES or institutions set up an infrastructure for the shared area able to serve concurrent connections from all the WNs of the computing farm or computing labs, it cannot be excluded that there will be peaks of load for the NFS/AFS server. This can be due to the concurrent execution of application software especially demanding for the file system hosting the software, like scripts with recursive listings of directory or code compilation and thus many system calls. These situations are difficult to foresee, as they might depend on the activity of private users, which is totally uncoordinated, and they are rather difficult to manage, since setups based on NFS/AFS servers are not easily scalable. Furthermore, the

Cloud environment will increase the latency to get the software from the NFS server due to this lost of performance.

Another very frequent source of issues resides in software installation jobs. Almost daily some of these jobs fail either because there is no space left on the shared area, or because the path to the shared area is not correctly set, or because network connection is too slow and the execution exceeds the job timeout. Also, permission problems have been often observed, i.e. the job has no permission to write on the shared area, due to some lock on the NFS file system.

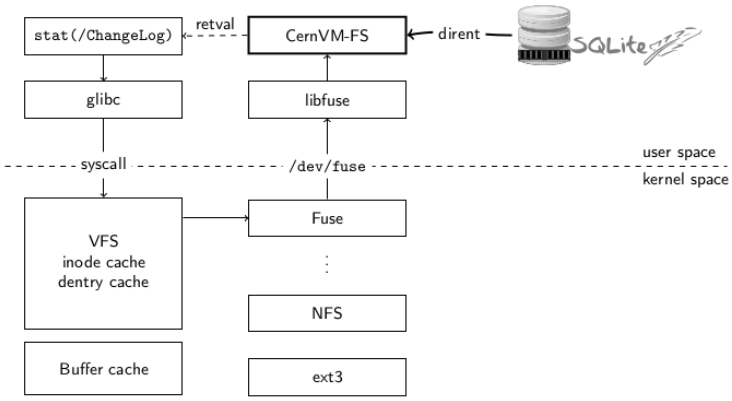


Figure 3.1: CernVM-FS internal structure.

One possible solution could be the *CernVM-FS software*, which is a network file system developed in the framework of the CERN virtual machine [29] project at CERN. According to this model the WNs access the software deployed in a central repository at CERN through HTTP protocol by mounting the CernVM-FS. CernVM-FS provides efficient caching functionality of files data and metadata, and reduces to the minimum the network latency when the software is accessed several times. During job execution, the WNs access the software through the HTTP protocol. Only the programs and libraries that are actually used during run time would be accessed, and then cached locally on the WN local disk. The optimized caching mechanism of CernVM-FS limits the network traffic between the node and the origin web server, which is furthermore reduced by mean of a zlib compression of the transferred files. Additionally, one or more HTTP proxies can be deployed at the SITE, in order to cache locally the software packages and to reduce the load on the central software repository at

CERN, as well as to reduce latency due to data transfer through the WAN. CernVM-FS can be mounted as a normal file system through File in the User Space (FUSE) [63]. Using FUSE, CernVM-FS has been implemented as a file system in user space as shows Figure 3.1, and it has been extensively tested in Cloud environments [20], with the virtual images of CernVM, and in physical platforms with different OS.

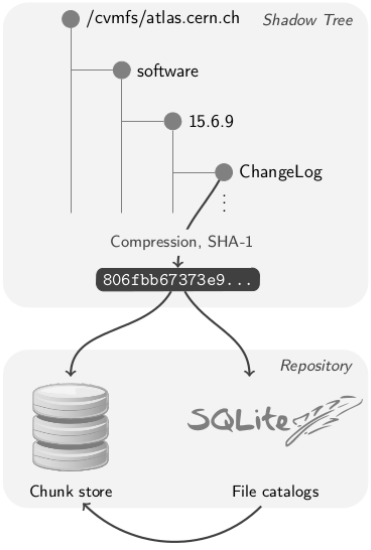


Figure 3.2: CernVM-FS shadow tree for Atlas experiment.

Clear advantages of this solution are the elimination of the complicated task of installing the software SITE by SITE, and the fact that the SITE does not need to maintain a shared area and mount it through some protocol in all WNs. The way CernVM-FS is working in World Large Hadron Grid is with an independent repository path for each LHC experiment as is shown in the Figure 3.2, as well as it has been designed to create a directory subtree in a read-only web server, so the client side only needs HTTP/HTTPS connectivity to the server.

CernVM-FS does aggressive file caching at different levels. The files are cached on the local disk of the computing nodes as well as on proxy servers, allowing the file system to scale to a very large number of clients. Furthermore, if some proxy fails the network traffic could use another available proxy, as shows the Figure 3.3. Thanks to that, CernVM-FS solves some problems that are present in other software distribution systems such as NFS [86] [18], which overloads the shared areas when there are millions of system calls in few minutes.

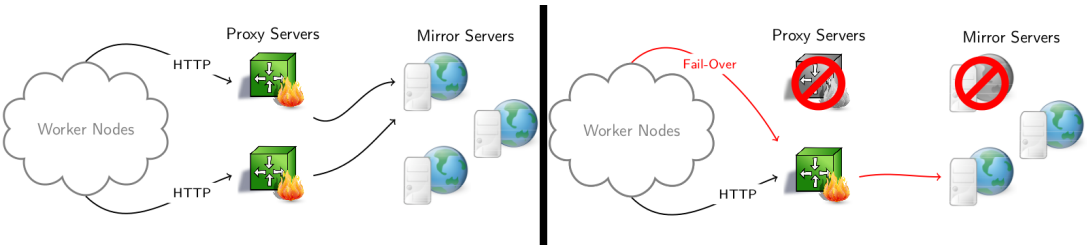


Figure 3.3: CernVM-FS proxy and mirroring errors.

In the software cycle on CernVM-FS, the procedure to construct, install and validate software versions is the responsibility of a version manager. Once this is done, the directory subtree has to be recreated in the repository. In the current version, the repository has a particular format, which is a content addressable storage called Shadow tree. From the next version forward a new backend with snapshots will be used [17]. The creation of the repository includes the tasks of creating the file catalog(s), compressing the files and calculating content hashes. Moreover, the files are stored locally, within a cache on the server, as SHA1 data fragments. This is done in order to exploit the SHA1 redundancy, and to use it as a key when downloading files. This avoids possible firewall blocking of HTTP requests to download files named *root.exe*. Once this is done, the new software is released through the *CernVM-FS server*.

The native CernVM-FS of CERN repository has a configuration of the cache hierarchy as shown in Figure 3.4. The tests performed in this thesis have used the standard one deployed in the LHCb production environment. Some of the features of this structure as cached hierarchy are, the local batch node memory, the local batch node disk cache, the backbone is Stable and distributed and public p2p network of webservers, the Cloud SITE Squid proxy servers, the CERN distributed stratum CernVM-FS mirror servers, the CERN central repository and finally the worker nodes are independent P2P cells building a decentralized memory cache.

CernVM-FS offers several repositories for different experiments. Figure 3.5 shows in the of the left picture (A) some features of the experiments software such as the number of objects of each of them or the type of system call. And it shows the number of increasing objects, in a period of time, in the LHC experiments under the CERN CernVM-FS repository. For some of the tests described in this thesis, the lhcb.cern.ch repository has been used in order to run Monte Carlo LHCb applications.

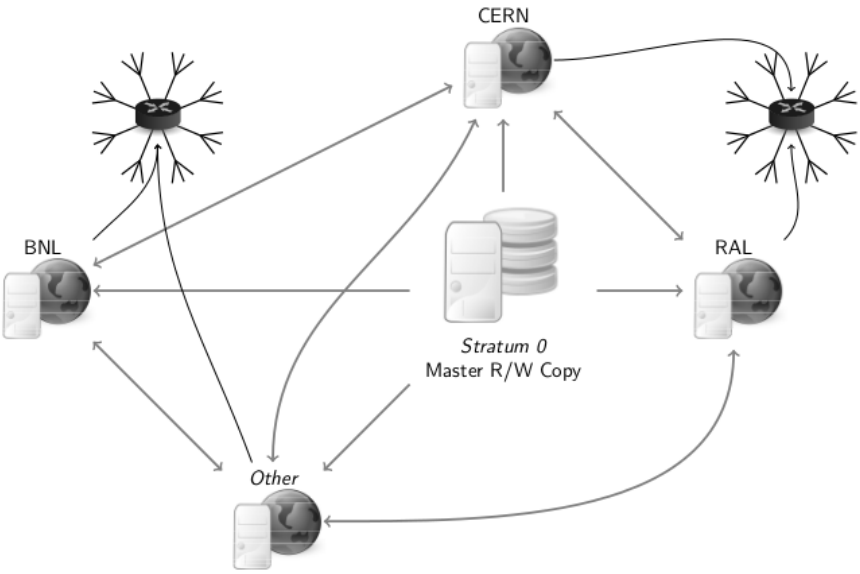


Figure 3.4: CernVM-FS stratum servers.

Some previously comparisons using LHCb software [87], NFS vs CernVM-FS, have shown that the job execution time is equal or lower than the one observed using the current setup based on a shared area accessed through NFS software. The network traffic seen between the WNs and HTTP proxy and between this and the central servers at CERN are modest, illustrating the efficiency of the local cache and data compression policies of CernVM-FS. Also, the size of the local disk cache needed for a realistic application has been found to be a small fraction of the size of the local disk available in a typical WN today. The test was running at PIC TIER-1 [107] over 16 WNs that were put off line from the computing farm and configured to be visible under a test queue. Each node is a 8 core system, running SL5 and gLite 3.2 WN middleware. The test consisted in executing a LHCb DaVinci test job, which is an LHCb package for data analysis, after setting the environment in order to read the software through the CernVM-FS. The results relative to the execution of this test job show a very low execution time for CernVM-FS, and no dependence with the increasing number of parallel jobs. On the other side, when reading the software from the shared area through NFS

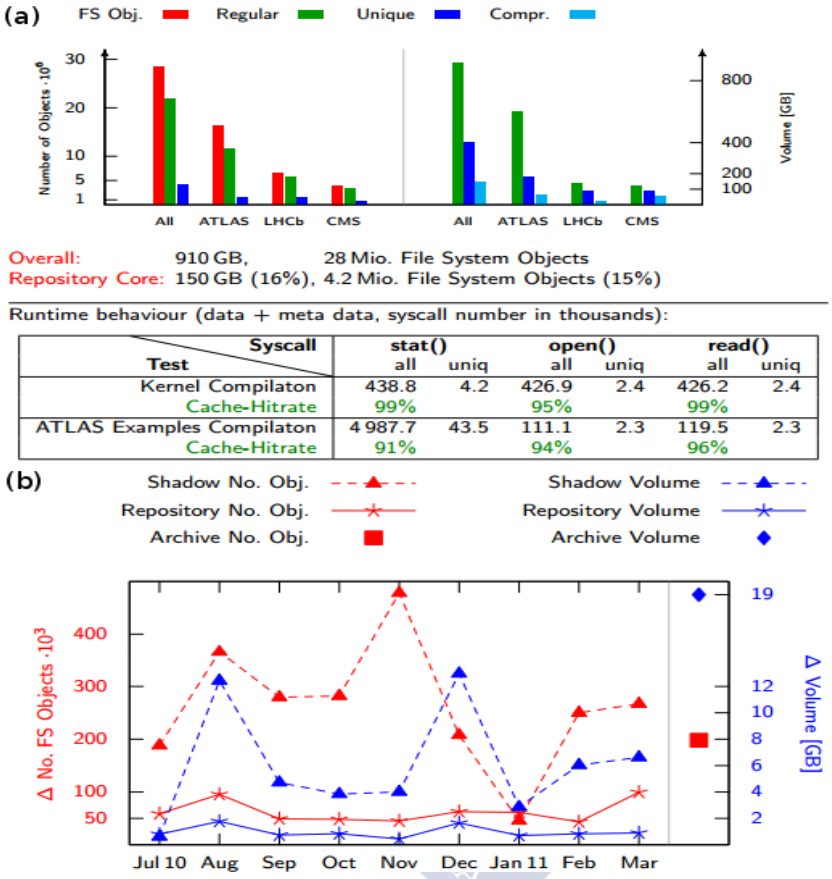


Figure 3.5: Features of the LHC experiments software.

protocol there is a clear dependence with the number of parallel jobs running on the same node. Several runs have been taken, when the load of the NFS server (quantified by the CPU percentage usage) was at different levels, from almost *0% to 80%*, depending on the load on the TIER-1 at that moment, since the production NFS server was used to do the measurements. It has been verified that the load of the NFS server has an effect on the client side, as the total number of jobs (128) is by far too low to overload the NFS server. Moreover the same behaviour has been observed submitting an increasing number of parallel jobs to only one node.

In order to compare with another way of software distribution, we have studied Tarball servers vs CernVM-FS. Next sections describe the process and results.

3.1.2 CernVM-FS vs Tarball Comparison

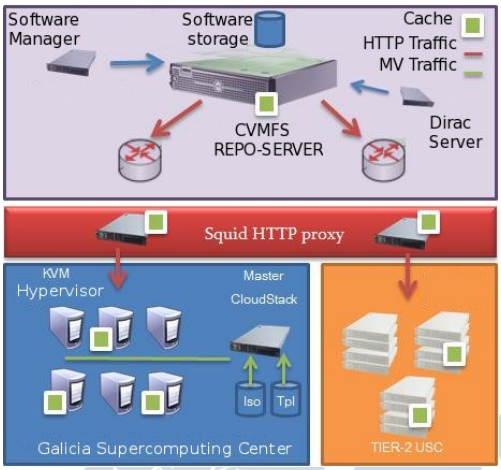


Figure 3.6: CernVM-FS repository and Architecture Deployment.

This subsection defines the deployment of CernVM-FS in order to have a test infrastructure as realistic as possible. An heterogeneous test environment has been deployed, consisting of virtualized nodes using CloudStack in the Galician Supercomputing Center (CESGA), and normal nodes in the LHCb TIER-2 cluster of the University of Santiago de Compostela (USC). The main components of the test infrastructure are, at the USC TIER-2, a CernVM-FS repository served by Tomcat on a machine of the LHCb cluster and a Squid HTTP proxy, which is important to check the scalability of the solution. An additional machine with a proxy server is also installed at CESGA to improve scaling. To test the possible overhead caused by CernVM-FS when executing typical HEP analysis jobs, a test-bed environment was deployed based on the ROOT system (http://root.cern.ch).

Figure 3.6 shows the repository deployment and the rest of the CernVM-FS architecture for testing the loss of performances in a worst case scenario for the CernVM-FS, which can then be compared with the direct download, as detailed below. Table 3.1 shows some of the

properties of CernVM-FS and the ROOT software, in particular, the number of files and the cache size.

Filesystem	**Size (MB)**	**Files**	**Avg. Size (KB)**
ROOT	218 (56 .tgz)	4630	2.5
CernVM-FS Cache	69	628	4.0

Table 3.1: File properties for ROOT and CernVM-FS.

Section 5.3 describes the algorithms and the benchmarks that were running in these deployment.

Benchmarks

The benchmark tests show a comparison of CernVM-FS versus Direct Download (TarBalls Files) from the same HTTP server. The VMs used are based on the KVM hypervisor, which is known to have about a 5% of performance reduction due to the virtualization overhead. To make a reliable comparison from a known and reproducible state, all caches are cleared up in every test iteration (the so called cold-cache method). After a specified number of iterations are performed, the one which provided the worst results is chosen. The aim is to measure the loss of performance in a realistic case where a typical user has never launched any job in the nodes beforehand. This seems the most realistic simulation because it can be compared as well with the direct downloads tests, where the user is not going to own the cached software when downloading the files.

The first algorithm in the Table 3.2 describes the application logic used to test the CernVM-FS performance and scalability in terms of execution time. The second algorithm shows the application logic used to test the Direct Download execution times in some scaling setups.

Each of the tests was performed several times to compare all the results and to verify the errors of network saturation or peak network, which could have affected them. It was taken into account the impact of the bandwidth in wide area networks, where there is a higher latency than in local area networks.

Results

The results of the benchmarks are shown in Figure 3.7, which on the left has a test using non virtualized nodes and on the right a test on a virtualized nodes environment. The comparison

Algorithm 1: **CernVM-FS Performances**	**Algorithm 2:** **Direct Download Performances**
Input: Node list Output: Time results if { multicore execution } then; run iterate in each node/core time A ? mount CernVM-FS set env time B ? run rf202 extendedmkfit if { node execution finish } then; get results() umount ? CernVM-FS delete ? Squid cache in each subnet delete ? CernVM-FS in each node return [worst result]	Input: Node list Output: Time results if { multicore execution } then; run iterate in each node/core time A ? download root set env time B ? run rf202 extendedmkfit if { node execution finish } then; get results() delete ? Squid cache in each subnet return [worst result]

Table 3.2: Testing algorithms of the software area.

between virtualized and non virtualized nodes has no sense, since they are heterogeneous environments, with different computing power, different network bandwidth, and additionally, the non virtualized results are performed in a non dedicated production environment, while the virtualized is a dedicated environment.

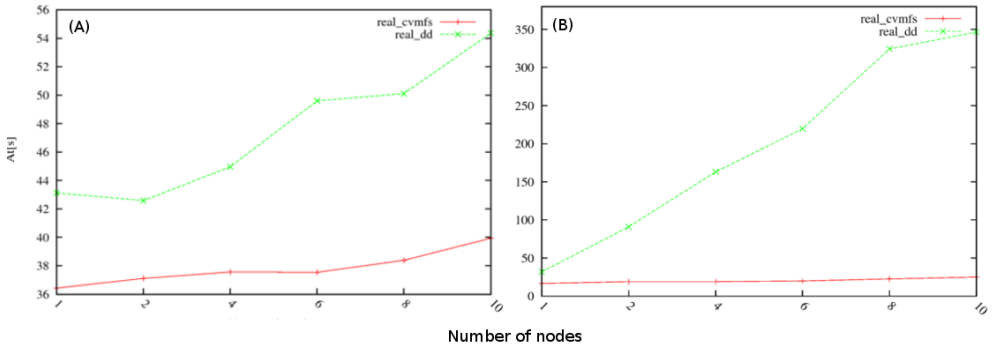


Figure 3.7: CernVM-FS comparison with Direct Download: (A) execution time of core/job in non virtualized nodes, (B) execution time of core/job in virtualized nodes.

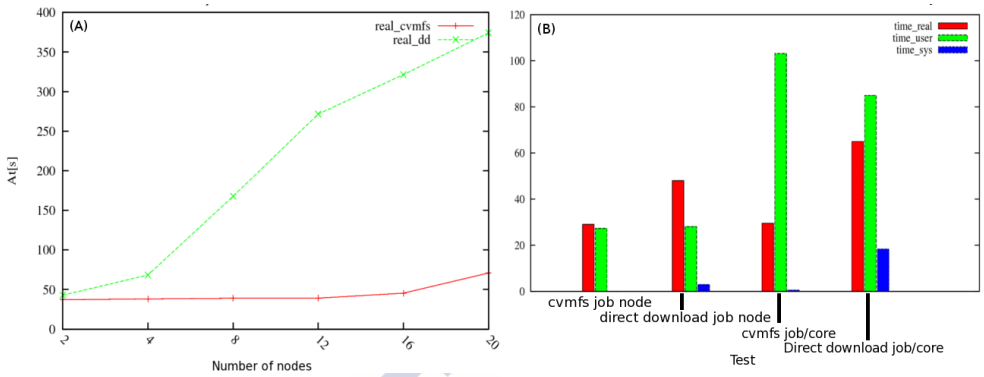


Figure 3.8: CernVM-FS Scalability on the Node Increasing and on the Remote Access: (A) Execution time of job/core in normal and virtualized nodes (B) Several types of software execution from USC CernVM-FS Repository to Barcelona University Cluster.

The left plot on Figure 3.7 shows the real execution time of tests in non virtualized nodes as a function of the number of nodes, for two different software repository infrastructures: CernVM-FS (red line) and direct HTTP download (green line). The execution time when using the CernVM-FS repository is substantially smaller than the one using direct HTTP download. The latter also increases strongly with the number of software requests, showing a clear advantage in favour of the CernVM-FS repository for a big number of simultaneous jobs running in the system. The low overhead of CernVM-FS caused by the software downloading needed to run the user's job could still be further reduced in a realistic situation, because usually that software has been already cached locally by previous jobs. This test (left plot) was performed in the same local area network, therefore there is not latency effect. The right plot of the Figure shows the execution time of tests in virtualized nodes. When using the HTTP download method (green curve) the time increase is proportional to the number of nodes, reaching about 6 minutes in the worst case (10 nodes). On the other hand, the curve corresponding to the CMVFS solution shows no significant execution time increase.

The left plot on Figure 3.8 shows the tests execution time in environments composed of a mixture of normal and virtual nodes. Again, for the the HTTP download method the time increase is proportional to the number of nodes involved in the test, while the CernVM-FS shows only a very moderate increase when going from 16 to 20 nodes. This time increase is due to network congestion. The right plot of the same figure shows the execution time (one job per node and one job per core) for four different tests. In this case, the real time

(red colour) increase is due to the latency of wide area networks, and it roughly doubles the execution time when going from CernVM-FS to HTTP download.

Summarizing the results obtained in the PIC tests on NFS and CernVM-FS comparisons, and the tests on Tarball servers and CernVM-FS comparisons, it is concluded that the best scalable and reliable software distribution system to be utilized in this thesis is CernVM-FS.

3.1.3 Software Service Repository

This subsection proposes a **CernVM-FS service deployment**, integrating different software appliances to allow the adoption in the different SITES such as universities, colleges or schools that have high computational resources in their computer labs or cluster farms with different setups. First the CernVM-FS software repository deployment is described. Second, it shows how an integration script can be run by the users so, depending on the SITE where the job run, a native CernVM-FS client or, alternatively, a user space CernVM-FS client with the proper configuration, is dynamically set up to run on the given SITE. And finally, are shown the results of benchmarks of this repository service.

Deployment

The CernVM-FS server repository was deployed at University of Santiago de Compostela, with the purpose of running the integration test, as well as the performance tests of the possible alternatives. The production CernVM-FS server repository service would be installed, configured and supported at CESGA, as a contribution to the Ibergrid operations [49]. From the experience of the LHC experiments, which have in production a CernVM-FS repository with a specific deployment for the Tier-1 SITES, it is clear that the scaling issue of the server should be considered. Initially, a single central CernVM-FS server can be deployed, but previous experiences have shown that it is necessary further server deployment to scale up even with a not so large user community or software repository. For example, the LHCb operations in 2011 shows that about 3,000 sustained jobs running in the early SITE adopters of CernVM-FS (Tier-0, Tier-1 and Tier-2 SITES) may overload a single CernVM-FS server, causing performance degradations on the server side. A Squid proxy is deployed at the SITE of the CernVM-FS server to support those client requests that have not a a local Squid proxy. Our CernVM-FS server scaling have to face similar situations, therefore two server stages can be useful, following the learned lessons of the LHC experiments. In these sense, different server configurations were deployed, first a stand alone CernVM-FS server or dedicated

server, other option is a central master CernVM-FS server not directly accessible by clients, with a set of associated mirrors distributed along the infrastructure (the so called stratum CernVM-FS servers). And finally a small number of mirrored CernVM-FS servers with a round robin scheme [16].

For a user to run a job using the CernVM-FS repository, the worker node must be running a CernVM-FS client, which should have been installed and configured previously by a privileged user. Unfortunately, this is not the case for many of the WN of SITES or the computers in institutions. A possible solution for this problem could be the use of statically compiled versions of CernVM-FS and Parrot. We propose to supply a CernVM-FS client integration script. This script will check if a native CernVM-FS client is installed in a specific WN and, if that is not the case, it installs and configures a static binary of CernVM-FS and Parrot to be run in the user space. This is not the most desirable option, because the local cache on CernVM-FS client instance would not take advantage of the WN local caching, since the caching will be done at the user level, not at the WN level as in a native CernVM-FS client instance.

Parrot is an open source software that can be used as an application environment for data-intensive computing, acting as an interposition agent. It can be used to attach existing applications to filesystems without requiring code or kernel changes. Parrot intercepts the system calls of an application through the *ptrace debugging interface*: system calls that do not affect I/O are passed through unmodified, while those affecting I/O, like open or read, are executed by Parrot on behalf of the application. This allows Parrot to contact a remote storage resource like CernVM-FS, get the necessary files, and pass them to the application simulating the access to a local file. Parrot allows to get software from CernVM-FS repository with the help of the CernVM-FS 2.0 release library interface [30]. Parrot can be modified in order to compile with static libraries and to create the multi-platform static binary of the Parrot software within the CernVM-FS driver.

The integration of CernVM-FS and Parrot needs several libraries as *jemalloc, libcurl, libfuse* and *c-ares* in order to have the same functionalities that a CernVM-FS client. The main functionalities supported are intermediate web proxies and local disk caching for scalability. On the security front, data integrity is verified with cryptographic checksums.

The Parrot software can be configured using command line options or with environment variables. The main option is the remote repository that is set via the "-r" switch or with "PARROT_CVMFS_REPO" environment variable, to indicate the repository. The repository syntax is:

repo name:options repo name2:option2

In the infrastructure case, the options that have to be configured are "url", with the URL of the CernVM-FS server (url1;url2) and "proxies" contains the HTTP proxy list, such as (proxy1|proxy2) (default values are given by the -P option). Proxies separated by "|" are randomly chosen for load balancing. Groups of proxies separated by ";" may be specified for failover: if the first group fails, the second group is used, and so on down the chain. Other options to be configured are: "cachedir", to indicate where to store disk cache; "pubkey", with the public RSA key that is used to verify the whitelist signature; and "mountpoint", with the root path of the repository.

The main disadvantage of Parrot with CernVM-FS client library is the limitation of only one repository mounted at a time. Parrot can destruct and re-initialize libCVMFS when switching between repositories due to severe performance penalty for frequent switching.

Besides Squid caching is used between the CernVM-FS server side and the CernVM-FS client, which is another caching layer that uses the simple HTTP proxy service. Squid caching aims to improve performance, thanks to user affinity using a common local CernVM-FS proxy server, and also to reduce the CernVM-FS server load. Currently most of the SITES have a Squid server to provide HTTP proxy in the working nodes. For those SITES which are not able to support HTTP proxy for CernVM-FS caching, additional remote Squid proxies will be deployed, which give worse performance than local Squid, but are better than no proxy. For those SITES having a CernVM-FS client installed on the working nodes, this information of the associated HTTP proxy is configured by the administrators. For the rest of the SITES, the client integration script checks if an environmental variable at the WNs (called "CVMFS_HTPP_PROXY") is defining a list of local HTTP proxies for CernVM-FS. If this variable has been not defined by the administrators of the SITES, the client integration script will define it with a set of default remote Squid proxies.

Users should be able to get their software from CernVM-FS repositories accessing to the software area repository, using the infrastructure proposed in this thesis, and this process should be scalable and fail-tolerant. This structure should be independent of the SITES that execute their jobs (i.e. the SITES may have installed or not CernVM-FS clients), and it should be able to distribute the software of the VOs, providing the tools and the mechanism to get the software independently of the SITE where the job would execute. This thesis proposes a software repository with several Squid caching level in the middlelayers between the SITES where the software is being executed and the end-point, that is the software repository in

charge of distributing the software. This structure can be complemented with SITE proxies in order to decrease latencies. The software repository should be split with the software of each VO, and it should have a downloadable script with preconfigured variables for the local execution of CernVM-FS-Parrot binary in the SITE.

The structure of the software repository has to follow the convention of CernVM-FS with the name of the VO group and, at the end of VO path, the script with the Parrot and the libCernVM-FS previously configured. This way, the software can be automatically obtained from the SITE.

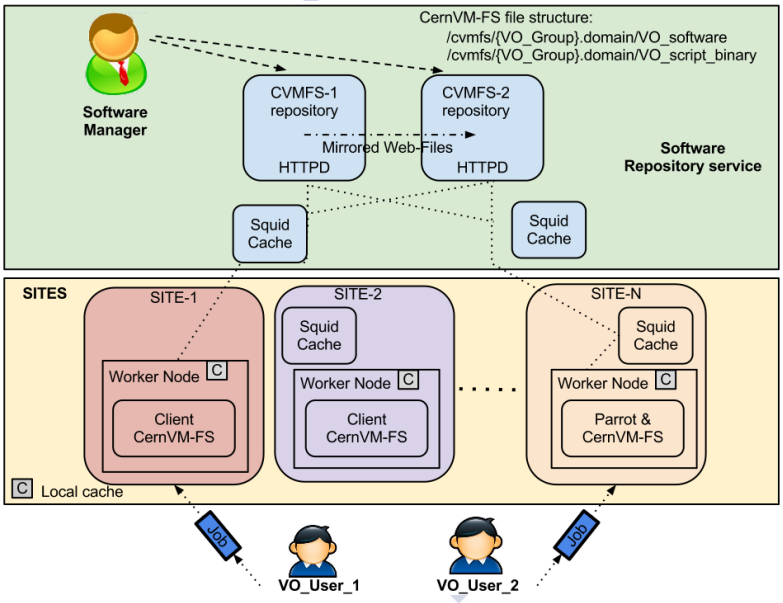


Figure 3.9: Ibergrid CernVM-FS software repository structure.

Figure 3.9 shows the steps of a user job that has been submitted to the infrastructure, which are:

1. Check if the CernVM-FS client has been installed in the node. If it is installed, the software can be obtained using the normal way;

2. In the negative case, the script with the Parrot binary and the environment configuration should be obtained from the repository. The script has the variable "PARROT_CVMFS_REPO" configured to access to the VO software;

3. Finally, the job should execute Parrot, specifying a mount point into the user space (e.g. /expSoftware). This mount point has to be indicated with the environment variable "VO_NAME_CVMFS_DIR". The downloaded files are cached in the /cache CernVM-FS folder.

The Software Repository Service is proposed as the Ibergrid software repository, and will be deployed in this thesis in order to test the infrastructure in Cloud environments. The Parrot binary will not be used because the infrastructure has a CernVM-FS repository available.

Prototype Results

This subsection summarizes the experience with the setup described in Section 3.1.2. It covers several typical use cases in the LHCb experiment and others, sending files of different sizes. This allows study of the stability of the setup. The infrastructure was created with an Cloud environment with the CloudStack v.2.2.3 manager, the server and client machines are virtualized. The host and the server are a Dell PowerEdge SC1425 with two processors PIV Xeon at 2.8 GHz and 1 GB of RAM per processor. Each one of the virtual machines has 500 MB of RAM memory and 10 GB of Hard Disk.

The test-bed has the following structure:

1. *A CernVM-FS repository served by Tomcat* on a machine of the LHCb cluster.

2. *A Squid HTTP proxy* is also installed in order to minimize response times. All nodes are interconnected with 100Mb/s switches.

3. The *CernVM-FS Clients* with the native software and the Parrot solution with CernVM-FS Lib.

Figure 3.10 shows the sum of the installation and configuration time with the CernVM-FS native client and the solution with Parrot and CernVM-FS lib.

In this test the Parrot client is faster than the native client, because the Parrot does not need to be installed and can be executed directly by the user once configured, while the native client needs to be installed by a privileged user.

Figure 3.11 shows two plots with the times needed to receive big and small files stored on the CernVM-FS repository to virtualized nodes. The left plot shows the time to receive 100 files of 100 MB each. The time is about 40 minutes with the Parrot solution, and only of 7

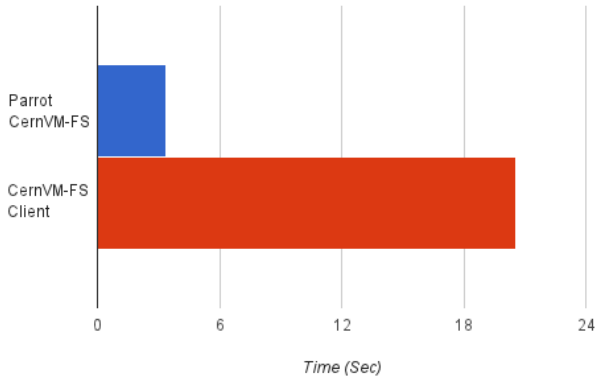


Figure 3.10: Sum of installation time and configuration time.

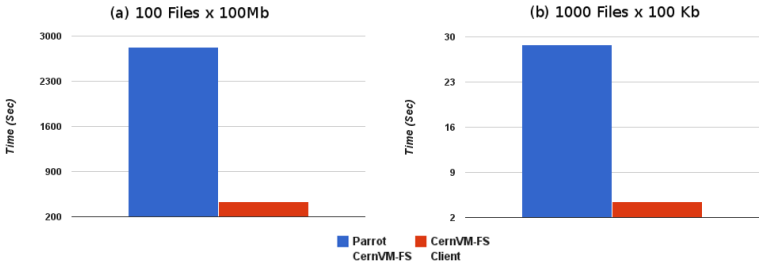


Figure 3.11: Transmission times for files of different sizes.

minutes for the native client. It can be seen that for big files, as well as for small ones (100 KB) shown in the right side of the plot, the native client is much faster than the Parrot one due to the fact that the Parrot ptrace debugging interface needs to catch the system calls. In these tests, the caching system has not proven useful, but in the next successive iterations, and on different nodes, both the proxy server and the local cache native system will influence the time to get the software.

Figure 3.12 shows a plot with the time needed to get files from the LHCb experiment on CERN Repositories, which are used to load several of the typical LHCb work environments

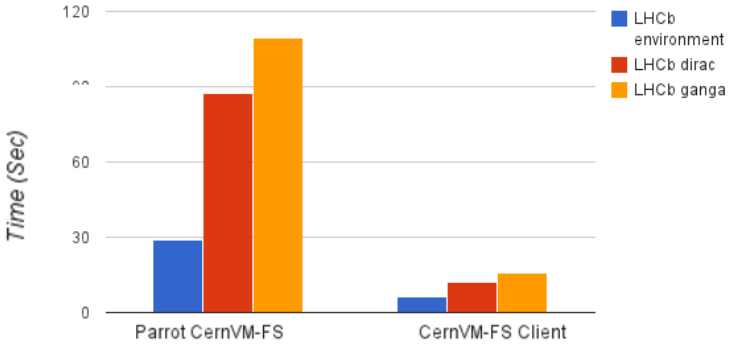


Figure 3.12: Times to load different LHCb environments.

with the SetupProject software, which has been introduced to address the need of a tool to prepare customized environments where to run LHCb applications. To be able to use the LHCb SetupProject utility, one needs a minimal LHCb environment. The first test is to load the minimum LHCb environment, which is shown in blue colour in the plot. The second test is to load LHCb DIRAC environment using the SetupProject utility. Finally the last test is to load the LHCb ganga environment [19]. The results show that the time to load environments using the native client is shorter than the one using the Parrot client, as was also the case for file transmission. These tests were repeated several times, and in some of them the Parrot solution was failing. When loading the LHCb minimum environment the failure rate was about 20%, with around 10 tests. When loading the LHCb DIRAC environment the failure rate was 40%. Finally, the failure rate observed when loading the LHCb ganga environment it was 60%. It is then clear that with large environments the Parrot solution is not reliable.

As a conclusion the Software Repository Service is based in previous tests and benchmarks and we have found an improvement in software distribution of CernVM-FS vs systems as NFS or HTTP servers. The thesis shows the way to get CernVM-FS software repository access by normal users, with improvements in the scalability, and avoiding problems as the high latency by using several level caching systems. The setup of the infrastructure shown as Parrot can be an option in very simple cases, but when the complexity increases Parrot

is unreliable. The described approaches make use of flexibility provided by CernVM-FS, with some characteristics as the multiplatform clients or the possibility of execution in Cloud environments, **for all these reasons the main conclusion is that the Software Repository Service with CernVM-FS is the optimal solution to provide software in the distribution environment of this infrastructure**.

3.2 Performance of High Energy Physics software

This section describes the experimental design and the hardware and software configurations used to run the HEP experiments in order to know if the execution of scientific software in the Cloud infrastructure has an reasonable performance [55]. Experimental design[1] is in general, the process of defining a method to gather and analyse data to study variable phenomena that can be under the full control of the experimenter or not. This method establishes a series of steps that will increase the possibility of obtaining a meaningful information from a experiment. The first step is the statement of the problem, that is, what is the question the experimenter wants to be answered. The second is the subject matter model or what are the factors that will affect the outcome of the experiment. The third step has three parts: treatment design (deciding which combinations of the values of the identified factors is the experimenter going to explore), error-control (how the treatments are assigned to the experimental units to minimize errors) and, the last part, sampling and observation (design how the data is taken). These steps define the experiment and the data to be taken. To analyse the data we first have to model the response, fourth step, in terms of the treatment and error-control factors. The fifth step is to choose the variable the experimenter is going to measure, or response. This choice is not always obvious but the variable should be chosen in a way that inferences and results from the experiment can be clearly stated an communicated. The last step is the principles of analysis or the application of statistical analysis to the data obtained in concordance with the experimental design and its associated model. Experimental design is, arguably, the most important part of an experiment because a faulty design could lead to the failure of said experiment.

At the treatment design phase it is decided how many factors should be used and how many levels per each factor. Often the treatments are chosen to have a factorial structure. In this case, we have several factors, each with a set of qualitative or quantitative levels.

[1] For a detailed exposition of the subject the reader is refereed to [79]

Suppose our factors denoted by F_i with $i = 1,\ldots,N$ being N the number of factors, and their respective levels by f_{ij}, where $j = 1,\ldots,\#\text{levels}(F_i)$. A factorial treatment is then a set of combinations of levels, one per factor, which are denoted by $(f_{1i_1} f_{2i_2} \cdots - f_{Ni_N})$. This type of treatment can be applied to any error-control design and it answers questions about the effects of single factors or the interference of several of the factors. Depending on whether all the factors have the same number of levels or not we distinguish between symmetrical or pure and asymmetrical or mixed factorial structures.

3.2.1 Defining Method and Experimental Set-up

Following the steps outlined in the introduction of this section we can describe our experiment as follows:

(i) **Statement of the problem**: our main objective is to study the performance of HEP scientific software on virtualized and non-virtualized environments. As aside questions we also want to find the behaviour of the applications in terms of the amount of memory/CPU resources available or the length of time the job is runing.

(ii) **Subject matter**: Considering the posed questions some of the factors to consider are obvious: virtualization, amount of memory, number of CPU cores, or length of the job. We also consider another factor, which we will name Benchmark, that will consider the software being executed.

(iii) **Three aspects of design**:

- **Treatment design**: we are going to use a mixed factorial design of the type $2^3 \times 3^2$. The 2 level factors are virtualizations whose values are baremetal and KVM, memory 2 and 4 GB and job length for which we will consider short and long jobs. The 3 level factors are the number of cores either 1, 2 or 3 and the benchmark.
- **Error-control design**: each combination is going to run four times with different random seeds to provide a reasonable amount of randomization.
- **Sampling and data taking**: the experiment will be run in a way that maximizes the use of the CPU resources. For example, for the configuration with two cores, there should be always two jobs running. In this case the number of jobs per combination ran will be 8: 4 repetitions times 2 jobs running simultaneously.

(iv) **Modeling the response**: we have four treatment factors: virtualization, memory, number of cores and length of the job; and one intrinsic factor: the benchmark used.

(v) **Response choice**: the variable we are going to measure is the walltime of the job.

(vi) **Principles of analysis**: to perform the statistical analysis we are going to use the multivariate analysis of variance or MANOVA for short.

The hardware used to run these tests consisted on two DELL Poweredge 1950 machines both with two Intel Xeon 5160 CPU's and 16 GB RAM. These processors have two cores each and run at 3 GHz. The operating system, where the tests were run, was Scientific Linux 6.3 with a gcc version 4.4.6. The software was served via a CernVM-FS [30] client, version 2.0.19-1.el6. Finally, with respect to the virtualization host, the KVM framework version 0.12.3+noroms-0ubuntu9.21 with a libvirt version 0.7.5-5ubuntu27.23 was used with Ubuntu 10.04 as the host operating system. To run the test in an environment as realistic as possible, the machines were kept fully loaded, running as many concurrent jobs as cores were available. The number of cores were controlled through the /sys/devices/system/cpu-/cpu#/online file, where # is the processor number in the system. The amount of memory accessible to the jobs was controlled via the cgroups service. A Python script was in charge of running the test, setting up the amount of memory at the job's disposal and, in the case of non-virtualized environment, configuring the number of active cores.

3.2.2 HEP-Software Hardware Architecture Performances Problems

As was mentioned in the previous subsection, our test bed was formed by machines with Intel processors. Nevertheless, those same tests were also run in AMD machines. In particular, two Supermicro AS 2022TG-HTRF equipped with two eight core AMD 6128 @2 GHz CPUs and 16 GB for the bare metal tests and 20 GB for the virtualized ones were used. A performance loss of around 5% was observed for FastJet and AIRES and for both, Intel and AMD architectures when comparing the performances in a virtualized and non-virtualized environments. In contrast the observed performance loss for GAUSS test was of 20-30% for AMD but of the order 5% for Intel. With the help of perf, a profiling program for Linux, the loss performance was traced to a specific function in GEANT4 software. We suspected that these differences were, in part, due to the fact that, whilst the version of GAUSS we used was precompiled by LHCb Computing group with some optimization flags, FastJet and AIRES were compiled by

us with no special optimization flags. The difference could be pin on KVM, since the versions shipped with Ubuntu 10.04 do not support all AMD CPU characteristics notably 3dnow and 3dnowext.

Because of incompatibilities with systems libraries, it was not possible to use a version of KVM more up to date. This same problem appears on other distributions like Scientific Linux, SUSE, etc. To confirm this last point the host operating system was updated to Ubuntu 12.04 which allowed to install a more recent version of KVM. The new version, along with NUMA, admit more AMD CPU characteristics, although still not all of them.

3.2.3 Statistical Performance Analysis

In this subsection the results of MANOVA analysis of the tests are presented. Around 800 simulations were executed with 28 different setups and hardware platforms. The KVM hypervisors were launched by hand with the different setups, and the data was extracted with a specific format. In the case of non-virtualized nodes, all setups were run with the same Python executable, which changed the different configuration of memory and cores, with a high isolation level.

Trend of Means and Median Set-ups

Once the jobs were executed in all the different setups, the resulting walltimes were selected from the data depending on the platforms where they were executed. Figure 3.13 shows the means of the walltimes for the different setups with virtualized (KVM) and non-virtualized or bare metal nodes (BM hereinafter). Two types of jobs are shown, large and short, with different configurations depending on the HEP-software.

The trend shows that the job walltime decreases when the number of cores increases. However, the amount of memory is not an important factor to run these type of jobs.

Figure 3.14 shows the walltime medians of each experiment. In the GAUSS medians, the interquartile ranges (rectangles) are bigger than for the Alice and Auger software, and the distance of medians between large and short jobs in FastJet and AIRES are shorter than the GAUSS, which shows a worse performance in GAUSS jobs with setups of 1 Core and 2-4 GB. Figure 3.14 also shows that in the case of GAUSS large jobs, the lower and upper quartile are more separated than in the case of short jobs, and it gives clear information about this type of jobs, which have variable performances depending on the number of events. In this sense, the setups of 2Cores-2GB and 1Core-4GB seem to have more stable performances.

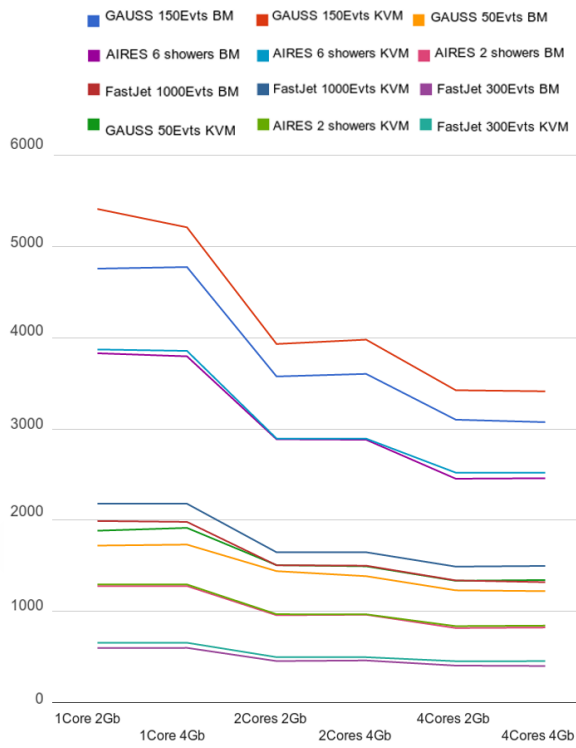


Figure 3.13: Walltime mean values of all HEP-Software in Virtualized and Non-virtualized nodes

BM and KVM Performance Analysis

This subsection, shows the results of Multivariate Analysis of Variance. The statistical design is a MANOVA 1-way, in which the dependent variables are the different HEP-Software with several setups of memory and cores, and with large and short jobs.

The survey of separate analysis of variance is shown on Table 3.3. In this case, the null hypothesis (H0) is the walltime of the executed jobs. All the dependent variables have statistic results that show a normal distribution, and they are significant with $P < 0.01$ and $F > 1$. The independent variable is the job walltime. The MANOVA with all the setups without means is shown in Figure 3.15.

The range of variables depending of discriminations setups are: FastJet KVM, FastJet BM, AIRES KVM, AIRES BM, GAUSS BM, GAUSS KVM.

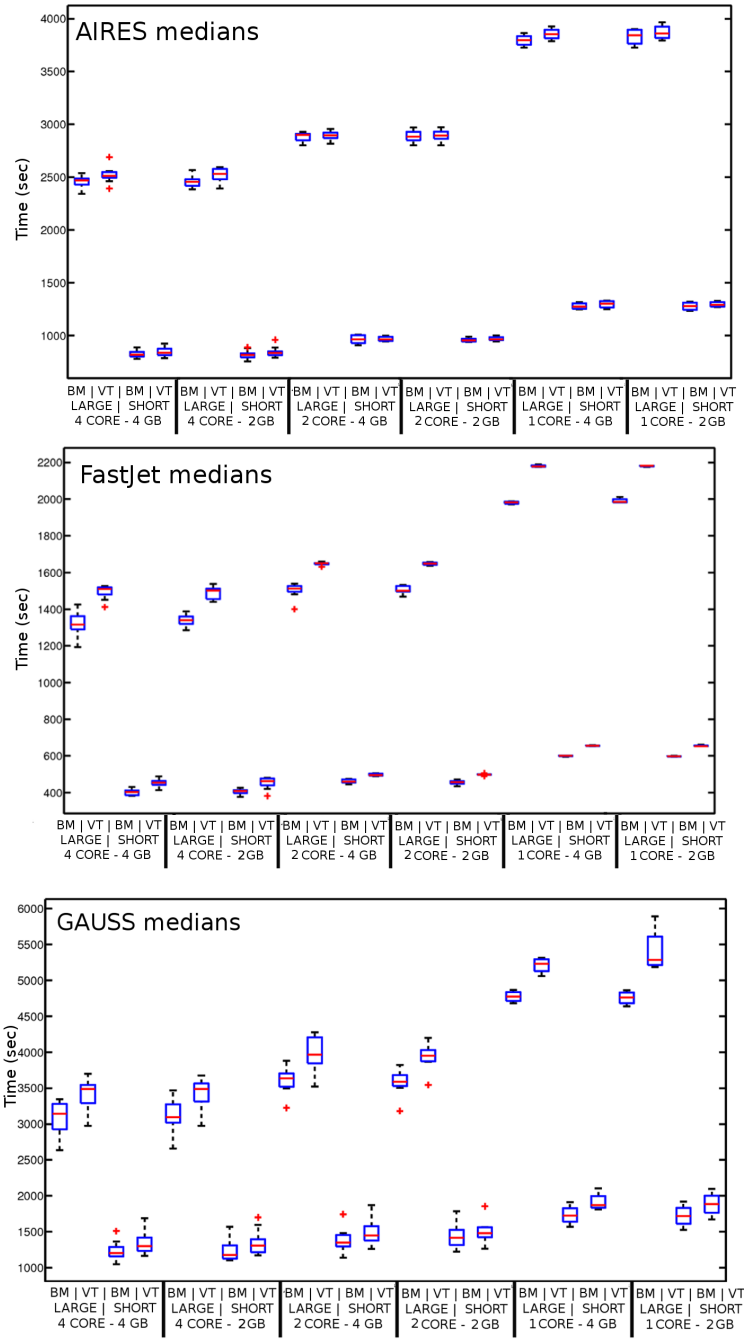


Figure 3.14: Medians of HEP-Software by experiment

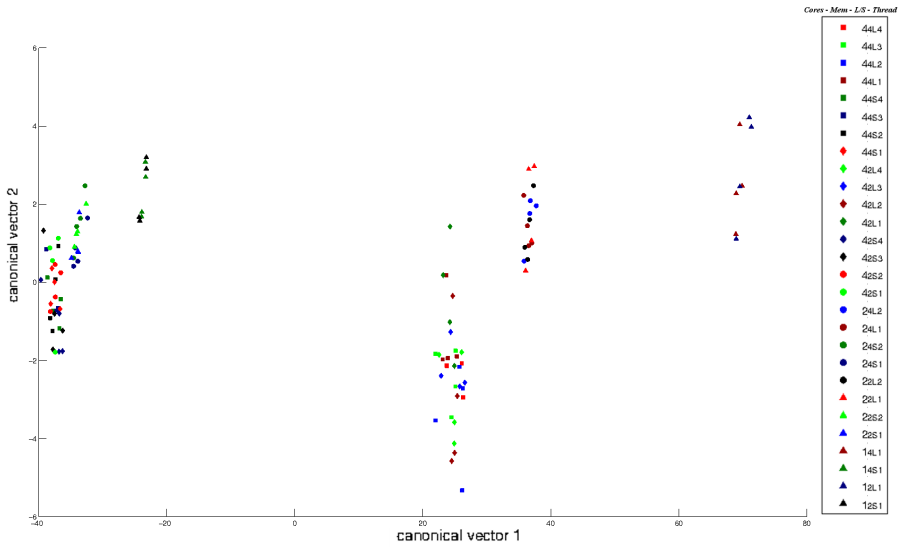


Figure 3.15: All MANOVA centroids. The legend indicates the number of cores, amount memory, large or short jobs and the thread number.

Table 3.4 is a MANOVA survey, to consider the possible correlations between the six dependent variables. The multiple analysis of variance offers results of all variables instead of the individual results of Table 3.3. The **Canonical Vectors** (CV) are projections of the original data in a lower dimensional space that allow a better discrimination between categories.

Table 3.3: Separate Analysis of Variance Survey

	GAUSS BM	GAUSS KVM
Snedecor F	157.9286	171.4093
Probability P	0.0000	0.0000
	AIRES BM	AIRES KVM
Snedecor F	2082.0955	1559.9788
Probability P	0.0000	0.0000
	FastJet BM	FastJet KVM
Snedecor F	1233.9208	2925.2098
Probability P	0.0000	0.0000

Table 3.4: MANOVA Survey

Exact Signif.	Wilk's Lambda	Prob. P	Snedecor F	Pillai's Trace
CV 1	5.143E-005	0	2.3368	2.5391
CV 2	0.08621	0	1.4718	1.5397
CV 3	0.4338	0.9324	0.7788	0.7384
CV 4	0.6143	0.9913	0.6235	0.4446
CV 5	0.7893	0.9985	0.4691	0.2229
CV 6	0.8979	0.9833	0.4443	0.1020

CV 1 and *CV 2* correspond to GAUSS, *CV 3* and *CV 4* to AIRES and, finally, *CV 5* and *CV 6* to FasJet. On each of the three groups the first and second canonical vectors correspond to BM and KVM respectively. In this case, the survey shows that the Wilk's Lambda and the Pillai's Trace, are both significant in terms of probability P and Snedecor F, but only for *CV 1* and *CV 2* of LHCb software. In these Wilk's Lambda is close to 0 (< 0.08) and the Pillai's Trace is higher than 1 with 2.53 *CV 1* and 1.53 *CV 2*. The conclusion is that in the MANOVA of all dependent variables the significant ones are the GAUSS performances in BM and KVM.

In this way, **simple ANOVA** is showing significance for each separated dependent variable and, as conclusion, the performances change depending on the setup. Furthermore, in the set of dependent variables in MANOVA is observed that there are correlations between the performances obtained for FastJet and AIRES and those obtained for GAUSS jobs, and this fact was expected as a response of the different software.

Further on the correlation analysis, we show a biplot in Figure 3.16. The biplot is the shadow in 2-dimensions of the normalized data, which maximizes variation. The measurements are shown as one unit of each axis, to give the idea of observations plotted correlations, which are corresponding to the distances between the points of the plot. The setup of the centroids is defined by core number, memory, large and short jobs, the thread number and finally, the Centroid number. This configuration is going to be the same for all the biplots in the remainder of the chapter.

Figure 3.16 plots the occurrences of the performances of the six dependent variables. The first analysis of the biplot considers the different setup discriminations. The main discrimination of correlated occurrences is the type, either large (right side) or short (left side) jobs. These discriminations are shown more in detail in Figure 3.17.

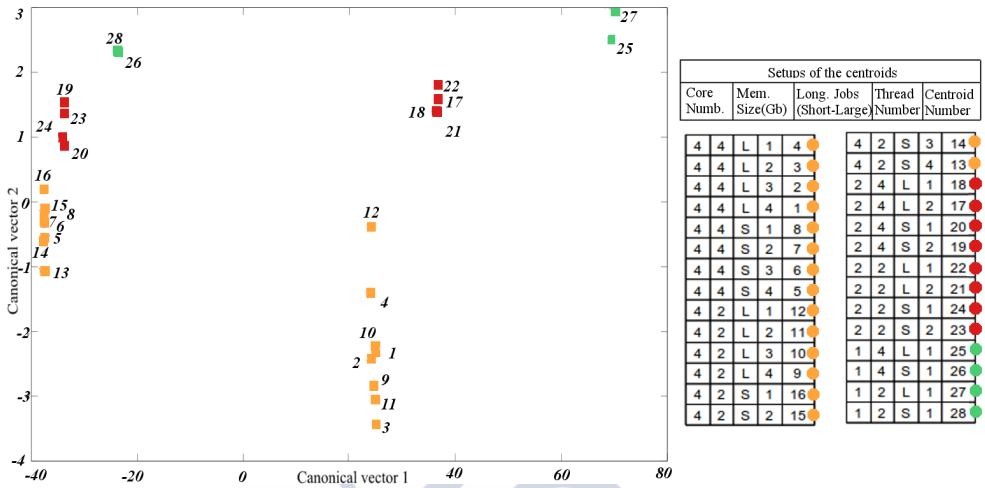


Figure 3.16: Correlation means of HEP-Software Wall-time

As shown in Figure 3.17, the occurrences of single cores (green color) are on similar areas with the setups of 2 and 4 GB, but the distance of the other setups along with the trend of the means, are suggesting that the single core setups are providing not optimal performances of HEP-Software. Another clear categorization are the occurrences of 2 Core setups (red color) in large and short jobs. Independently of the amount of memory, there are correlations between them. These correlations allow us to conclude that an increase in memory does not improve performances. The occurrences of 4 Core setups (yellow color) are correlated in short and large jobs, this categorization shows that in large jobs the occurrences are closest with 4 GB of memory, therefore this type of jobs are more stable.

Individually Performance Analysis of BM and KVM

Finally, in order to complete the Statistical Analysis, a study of BM and KVM performances was done. Table 3.5 shows the MANOVA survey that considers the possible correlations between three dependent variables in BM and KVM. And in this way, the set of dependent variables in MANOVA BM and KVM show a 2 dimensional significance with $P < 0.01$, (9.3296e-111, 0.002) in the case of BM and (2.1501e-118, 2.7345e-09) in the case of KVM. The *Pillai's Trace* is higher than 1 in both cases, and the *Wilk's Lambda* is close to 0.

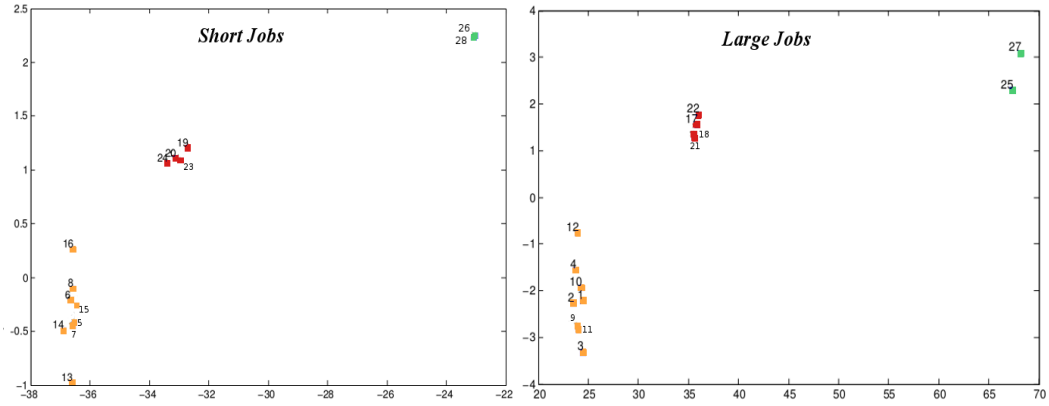


Figure 3.17: Centroids of short and large jobs

Table 3.5: MANOVA Survey

Exact Signif. BM	Wilk's Lambda	Prob. P	Pillai's Trace
CV 1	0.00035287	9.3296e-111	1.6816
CV 2	0.41196	0.002807	0.68244
CV 3	0.80721	0.72265	0.19279
Exact Signif. KVM	Wilk's Lambda	Prob. P	Pillai's Trace
CV 1	0.00023424	2.1501e-118	1.8774
CV 2	0.24353	2.7345e-09	0.8783
CV 3	0.8273	0.83787	0.1727

As in the previous case, the MANOVA analysis of all the dependent variables for the current case indicates that the significant variables are the performances of GAUSS in the BM and KVM environments. The rest of dependent variables are probably correlated with the ones that characterize GAUSS.

Figure 3.18 shows the centroids correlation of BM-MANOVA and KVM-MANOVA, they are similar to the sets of Figure 3.16. Notice that there are some differences such as the distance between large jobs, with setups of 4 Cores and 2-4 GB, in the case of KVM are

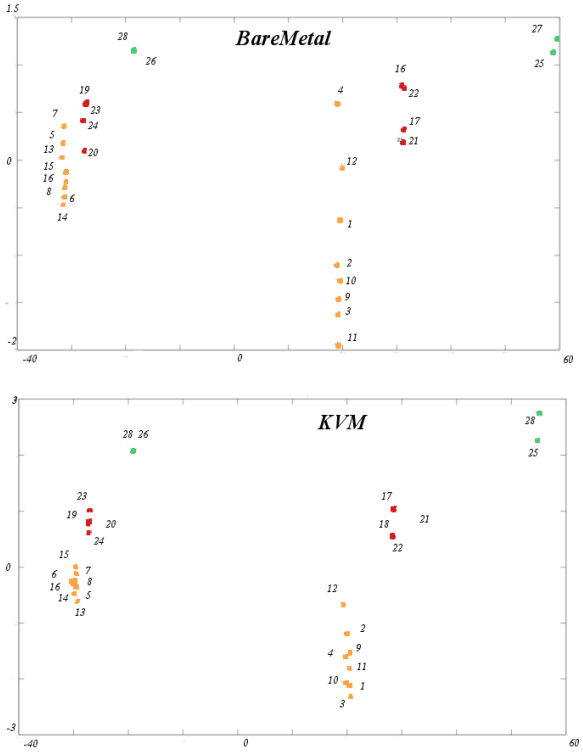


Figure 3.18: BM and KVM Centroids

closer than in the case of BM. This distance shows that the analysed HEP-Software seems to have stable performances in this type of setups for Virtualized environments.

An additional comment can be made about the distanced between the centroids in the KVM and BM separated analysis. These distances are higher in KVM for the different setups, which shows that in these HEP-Software the virtualized setup is a relevant factor to be considered when looking for a better performance.

The main conclusion of this section is that it is possible to create a Cloud infrastructure to be used by researchers to submit jobs of scientific data analysis.

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

CHAPTER 4

DESIGN

This chapter contains the software development methodology as the framework that is used to structure, plan, and control the developing process of software in this thesis.

4.1 Methodology

This section presents the methodology and the design approach chosen to solve the research problem that will be defined in the requirements section. Before focusing on the results, it is important to know the tools and techniques used during this phase. In particular, this section describes the Software prototyping process and the architecture design using this type of software lifecycle.

In every domain with a high level of uncertainty and, consequently in the domain of research, it is necessary to confront the requirements and the resulting conceptual design with state-of-the-art solutions drawn from the available literature. It is also necessary to conduct a feasibility study for every major step of the problem solving process. This results in a series of iterative steps that are performed as shown in Figure 4.1. At first, during the requirements analysis stage, brainstorming sessions are held and the research problem is analysed and defined in detail. Later, it is broken down into a number of smaller problems that are easier to understand and solve. Next, for each of these smaller problems, a detailed literature review is pursued. This may result in a list of potential solutions or at least hints and traces on how to solve the analysed problems.

As a solution to model these smaller problems, in this thesis The Unified Modeling Language (UML) was used, which is a standardized general-purpose modelling language in the

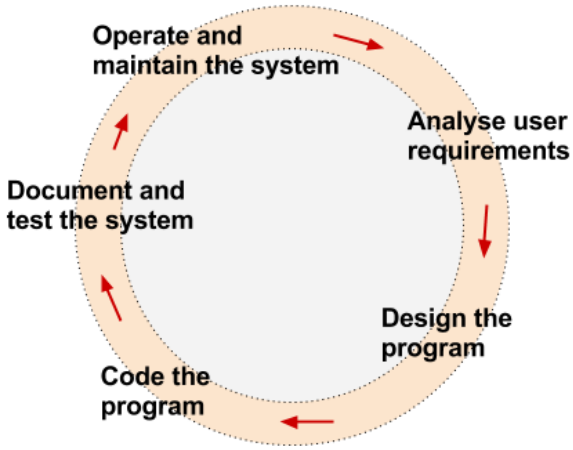


Figure 4.1: Methodology of software design cycle process.

field of software engineering [128]. The current UML standard contains several different diagrams and modelling techniques that can be applied at practically all steps of an IT system's design, implementation and deployment. In the requirements and design phase of the prototypes the following techniques were used:

1. **Use Case Diagram**: model the low level the infrastructure and the requirements.
2. **Activity Diagram**: model the multiEnd-Point virtual machine submission.
3. **Sequence Diagram**: depict the control flow during the job submission.
4. **Class Diagram**: model the classes of the project.
5. **Entity - Relationship model** (ER model): abstract way to describe the database.

Next sections will show some of the UML diagrams described.

4.1.1 Software Prototyping Process

Software Prototype [14] is an approach to software development that uses prototypes to help both the developers and their customers to visualize the proposed system and predict its properties in an iterative process. In the context of this thesis, the software prototype can be useful to create a first software to understand better the way as Cloud Computing works and how to create improvements in the integration of DIRAC and Cloud managers. In this way, the original purpose of the first prototype in this thesis is to allow users to evaluate developer proposals for the design of the eventual product by actually trying them out, rather than having to interpret and evaluate the design based on descriptions. In this sense, Prototyping can also be used by end users to describe and prove requirements that developers have not considered, and that can be a key factor in the final result and next prototypes. Interaction design in particular makes heavy use of prototyping with that goal.

This process contrasts with the 1960s and 1970s monolithic development cycle of building the entire program first and then working out any inconsistencies between design and implementation, which led to higher software costs and poor estimates of time and cost. The monolithic approach has been dubbed the *Slaying the Dragon* [96] technique, since it assumes that the software designer and developer is a single hero who has to slay the entire dragon alone. Prototyping can also avoid the great expense and difficulty of changing a finished software product. Figure 4.2 shows all the steps in the Prototyping Process

In this thesis the steps followed for the software prototyping process involve:

1. **Identify basic requirements**: determine basic requirements including the input and output information desired.

2. **Develop initial prototype**: the initial prototype is developed including basic functionalities in user interfaces and job submission to one Cloud manager.

3. **Review**: the users, examine the prototype and provide feedback on additions or changes.

4. **Revise and enhance the prototype**: using the feedback both the specifications and the prototype can be improved. Negotiation about what is within the scope of the solution may be necessary. With the changes introduced, iteration of steps 3 and 4 might be needed.

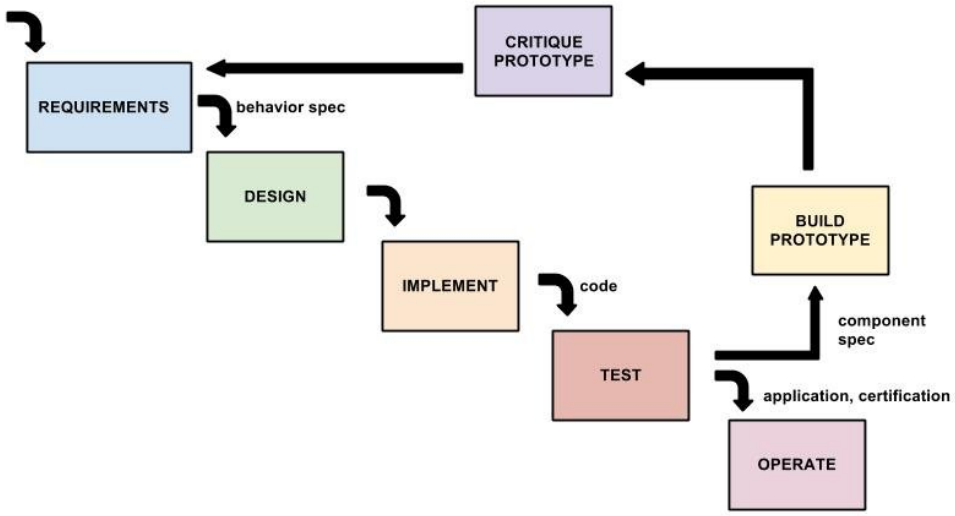


Figure 4.2: Methodology of software design, Prototyping Process.

4.2 Requirement Analysis

4.2.1 General objectives

The general objective of this thesis is to provide researchers an easy way to submit and execute their jobs to Cloud Computing environments. The system should be able to use Cloud resources in computer rooms of universities, institutes and public schools as was defined in Formiga-Cloud proyect, and the functionalities will be extended to be used in cluster farms. The integration of CloudStack manager and DIRAC is the starting point, for computer rooms virtualization. It also seeks interaction with other Cloud systems via the integration of OCCI interfaces and OpenNebula's Contextualization and with Big Data software over Cloud, as well as integration with the existing system, currently DIRAC management within EC2. Another objective is the complementarity of the new system created with the traditional works deliveries through the Web Portal.

The software described in this thesis is based in the Software Prototyping Process, where the creation of the final solution follows an incremental development. This new software should have the following features:

1. Manage Virtual Machines.
2. Manage Cloud Infrastructure with one Cloud manager and increase for multi-clouds.
3. Manage and configure the system for multiple user groups in a dynamic way.
4. Provide a way of deployment and configuration adapted to classrooms and clusters, being flexible, scalable, user friendly and dynamic.
5. Complement existing queuing system with job submission system to the Cloud infrastructure through Web-Portal.
6. Testing the infrastructure with real cases of multi-VOs.
7. Manage job submission to Big Data software.

The next section describes the Use Cases for these objectives.

4.2.2 User case analysis

This section defines the list of steps, typically defining interactions between a Role or Actor and Systems, to achieve the objectives. In this thesis there are a group of Roles and Actors that interact with the infrastructure:

1. **Cloud Admin** (A): in charge of manage the Cloud infrastructure.
2. **Researcher** (R): use the system and submit their jobs.
3. **DIRAC Admin** (D): in charge of manage the job submission platform.
4. **Software Manager** (S): in charge of manage users software.
5. **Big Data Admin** (B): in charge of manage Big Data software.

UC	Rol	Use Case
UC.1	**A**	Manage the Cloud infrastructure
UC.2	**A**	Upload virtual machines templates
UC.3	**A**	Manage the Cloud private network
UC.4	**R/D**	Manage the job submission for multi-VO
UC.5	**R/S**	Manage the user software for muti-VO
UC.6	**R/D**	Manage the job submission through Web Platform
UC.7	**R**	Apply to via Web Browser the access to the Platform
UC.8	**D**	Manage user data
UC.9	**D**	Monitoring job status and VM status
UC.10	**B**	Manage the Big Data software

Table 4.1: Use Cases of the infrastructure

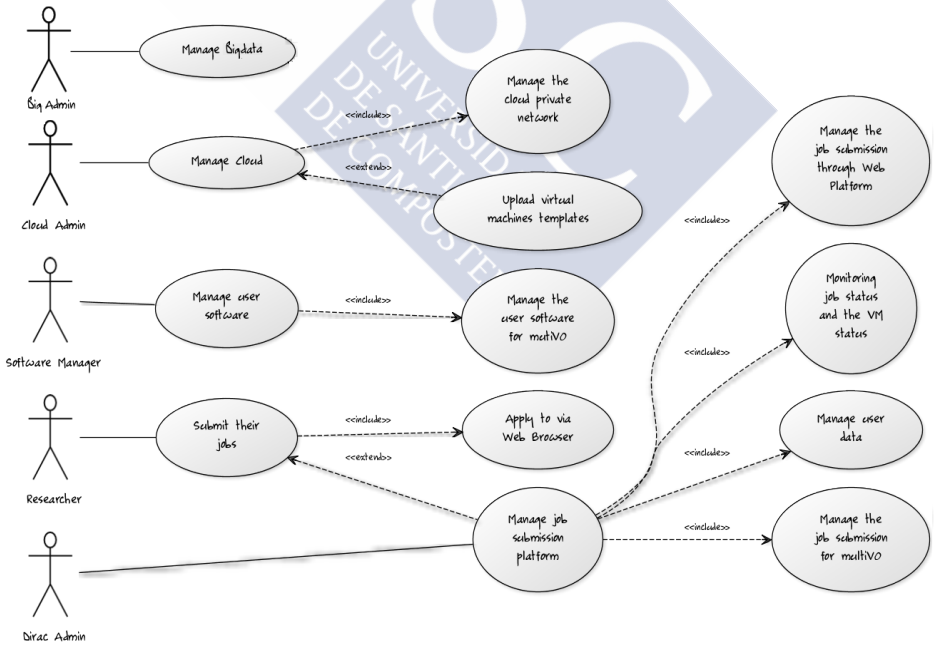


Figure 4.3: Use Case diagram with the system roles.

Role	A
Use Case	Manage the Cloud infrastructure
Acronym	MCI
Description	The Cloud Admin is in charge of the maintenance tasks of the Cloud infrastructure. He should take care of the integrity of the platform and the NFS storage.

Table 4.2: Manage the Cloud infrastructure Use Case

Role	A
Use Case	Upload virtual machines templates
Acronym	UVM
Description	The Cloud infrastructure needs templates, and the Cloud Admin is in charge of create and maintaining these templates.

Table 4.3: Upload virtual machines templates Use Case

Role	A
Use Case	Manage the Cloud private network
Acronym	MCP
Description	The Cloud infrastructure needs a private range network for the Virtual Machines submission and the Cloud Admin is in charge of assign this IP range.

Table 4.4: Manage the Cloud private network Use Case

Role	R/D
Use Case	Manage the job submission for multi-VO
Acronym	MUM
Description	The DIRAC Admin should take care of the DIRAC Infrastructure, furthermore he is in charge of the multi-VO user configuration, which is needed for users can execute their jobs. The DIRAC Admin have to manage the Cloud settings in order to submit the jobs.

Table 4.5: Manage the job submission for multi-VO Use Case

Role	R/S
Use Case	Manage the user software for muti-VO
Acronym	MUS
Description	The Software Manager is in charge of the maintenance of the software version for the groups that are being supported.

Table 4.6: Manage the user software for muti-VO Use Case

Role	R/D
Use Case	Manage the job submission through Web Platform
Acronym	MJS
Description	The DIRAC Admin is in charge of the setting up and maintenance of Web Platform. The Users or Researches submit their jobs through the Web Page.

Table 4.7: Manage the job submission through Web Platform Use Case

Role	R
Use Case	Submit a form through the Web in order get access of the platform
Acronym	AWP
Description	The Researcher could submit his acceptance in the infrastructure through the DIRAC Web Page.

Table 4.8: Apply to via Web Browser the access to the Platform Use Case

Role	D
Use Case	Manage User Data
Acronym	MUD
Description	The DIRAC Admin is in charge of the user data stored in the DIRAC main server.

Table 4.9: Manage User Data Use Case

Role	D
Use Case	Monitoring job status and VM status
Acronym	MJV
Description	The DIRAC Admin is in charge of the Virtual Machine monitoring and the user jobs, and he should take action in case of failures.

Table 4.10: Monitoring job status and the VM Use Case

Role	B
Use Case	Manage the Big Data infrastructure
Acronym	MBD
Description	The Big Data Admin is in charge of the maintenance tasks of the Big Data infrastructure. He should take care of the integrity of the platform and the Hadoop File System.

Table 4.11: Manage the Big Data software Use Case

The specific Use Cases for the roles are shown in Table 4.1. Figure 4.3 contains the Use Cases diagram. Use Cases are described from Table 4.11 to Table 4.10. Functional requirements needed in the Cloud infrastructure shown in Table 4.12 and Table 4.13 shows the traceability matrix.

There are other requirements that must be detailed such as packaging, support or license holder. These details are provided in the complementary specification, and they determine all things not covered in the Use Cases. Some of these requirements are:

1. **Easy management**: the system should have a user-friendly intuitive interface, which should help the researcher to easily get the results of the jobs.

2. **Reliability**: the system must be precise in the operations, and the infrastructure should be scalable in all components.

	Description	Comments
FR.1	The system allow to manage the Cloud infrastructure	
FR.2	The system should provide a mechanism that allows the user to upload an image of a virtual machine	The user should register his own image of VM
FR.3	The system should provide external access to the private Cloud	
FR.4	The system allow to get scientific software from scalable repositories	The software repositories could be external and the groups Cloud have their own repository which could include additional software.
FR.5	The software repository should have a specific format for the software groups	The specific path depends on the user group
FR.6	The system allows to job submission through the Web Portal	This will be possible with the proxy certificate delegation
FR.7	The users could registered through the Web Portal	This make the aceptance process more agile
FR.8	The software allow to dynamic management to create new instances for different user groups	This requirement will be an extension in the prototypes and will be necessary the Cloud manager API
FR.9	The system store the jobs of all users	
FR.10	The system save the history of the relation of Virtual Machines and SITES or EndPoints	
FR.11	From the system is possible the job monitoring and the queue management of the Virtual Machines	
FR.12	The system allow to manage the Big Data infrastructure	

Table 4.12: Functional requirements needed in the Cloud and Big Data infrastructure for job submission.

	MCI	UVM	MCP	MUM	MUS	MJS	AWP	MUD	MJV	MBD
FR.1	■									
FR.2		■								
FR.3			■							
FR.4					■					
FR.5					■					
FR.6						■				
FR.7							■			
FR.8				■						
FR.9								■		
FR.10									■	
FR.11									■	
FR.12										■

Table 4.13: Use Cases and functional requirements traceability matrix

3. **Development restrictions**: It will be developed in an OO language. On the other hand Web-Services (SOAP) are needed to Cloud Manager connections.

4. **Open-source development**: the source code will be developed as far as possible with Open-source tools such as DIRAC, CloudStack, Hadoop or MySQL. The resulting developments will be made available via integration into the central repository of the corresponding software package.

4.3 Architecture Design

4.3.1 Prototypes modelling

The previous subsection introduced the steps of software prototyping model, and this subsection shows the models and diagrams. Part (a) of Figure 4.4 shows the Activity diagram with all the components of the thesis in a Top Level of abstraction before focusing on specific more concrete scenarios.

Part (a) of Figure 4.4 shows that the researcher has 3 ways/interfaces to submit a job such as the web portal, the user interface or command line, and the DIRAC API. When the job is submitted to the DIRAC Server, the system should connect with the specific Cloud manager and then create the necessary instance through the Cloud API. The next step is to download

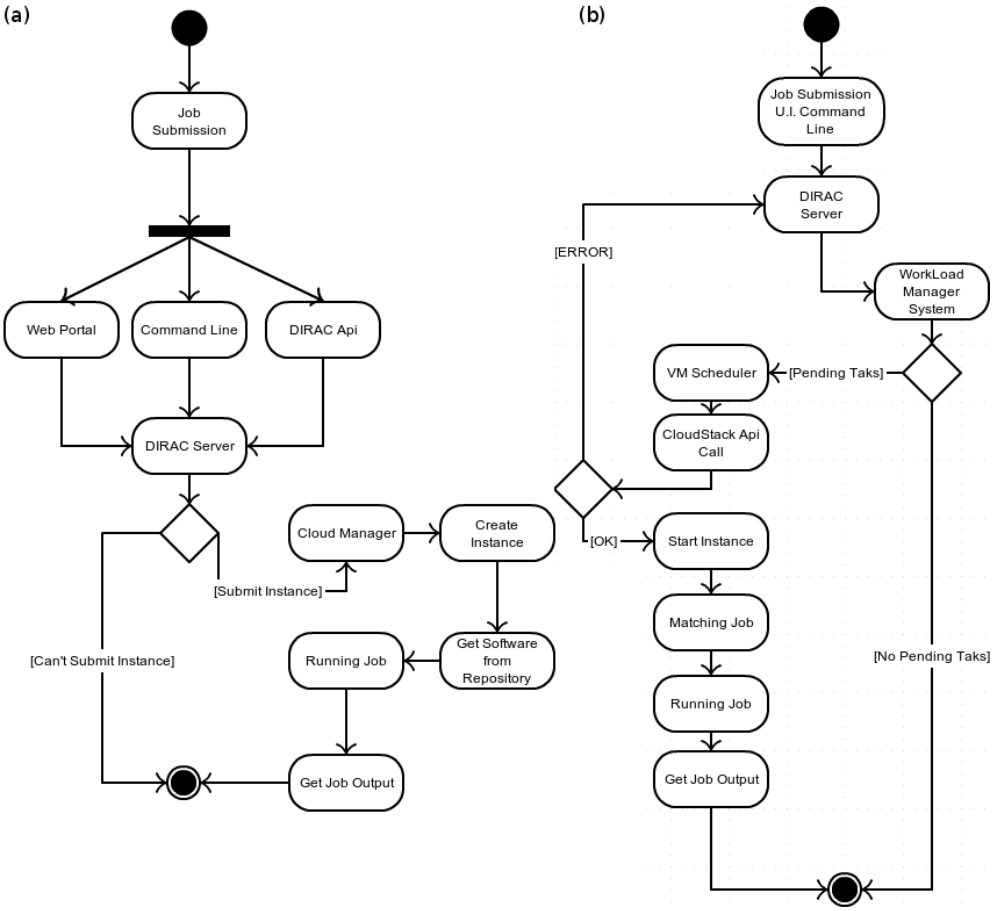


Figure 4.4: (a)Top level abstraction Activity diagram.(b)Job submission to CloudStack manager Activity diagram.

the software, and execute the job. Finally, the DIRAC Agent will execute the job, and upload the Output to the DIRAC Server. These are the steps and subsystems of the infrastructure with a high abstraction level.

Our first prototype or CloudStack prototype, is a basic Cloud Manager connection and job submission. This prototype is designed for Formiga-Cloud proyect, and it has provided feedback for the Multi-EndPoint prototype. Part (b) of Figure 4.4. shows the prototype,with the following steps:

1. Researchers send their jobs through the DIRAC User Interface to the *DIRAC's Worload-ManagerSystem* (WMS). The WMS is responsible for matching requests and create new tasks pending for execution.

2. The WMS checks if there are pending tasks, and if there are any, the *VM Scheduler* component gets the list of nodes in "Up" state and their processing load. Then, it compares the number of pending tasks and checks whether the available resources are enough. If the resources are not enough, it checks if there are Hosts with low processing load and, if this is the case, switch on more VMs.

3. In the phase of "Start Instance", after the standard boot sequence of the operating system, Scientific Linux 5 in this case, up-to-date Certification Authorities publickeys (CAs) and associated Certificate Revocation Lists (CRLs) are downloaded. A private *DIRAC slave configuration server* is started and it is configured to connect to the central configuration server to get the latest version of the configuration data. It remains synchronized with the master during the lifetime of the VM.

4. Afterwards, a *DIRAC Job Agent* is started. It reports back to the *VM Manager* the availability of the new VM. Finally a job is matched and run in the new VM.

In the second Multi-EndPoint prototype more functionalities are included. Furthermore, the design was done to easily support additional Cloud managers and different versions of them. The researches could submit their scientific jobs through the DIRAC Web Portal with the previous proxy delegation to DIRAC Server. The user jobs of several groups could be submitted by researchers in the Multi-EndPoint prototype through the Web Portal. Then, the computing node will get the specific software for the particular job belonging to the specific group. The node will execute this job on the platform requested by this group. Figure 4.5 shows the Sequence diagram of the process.

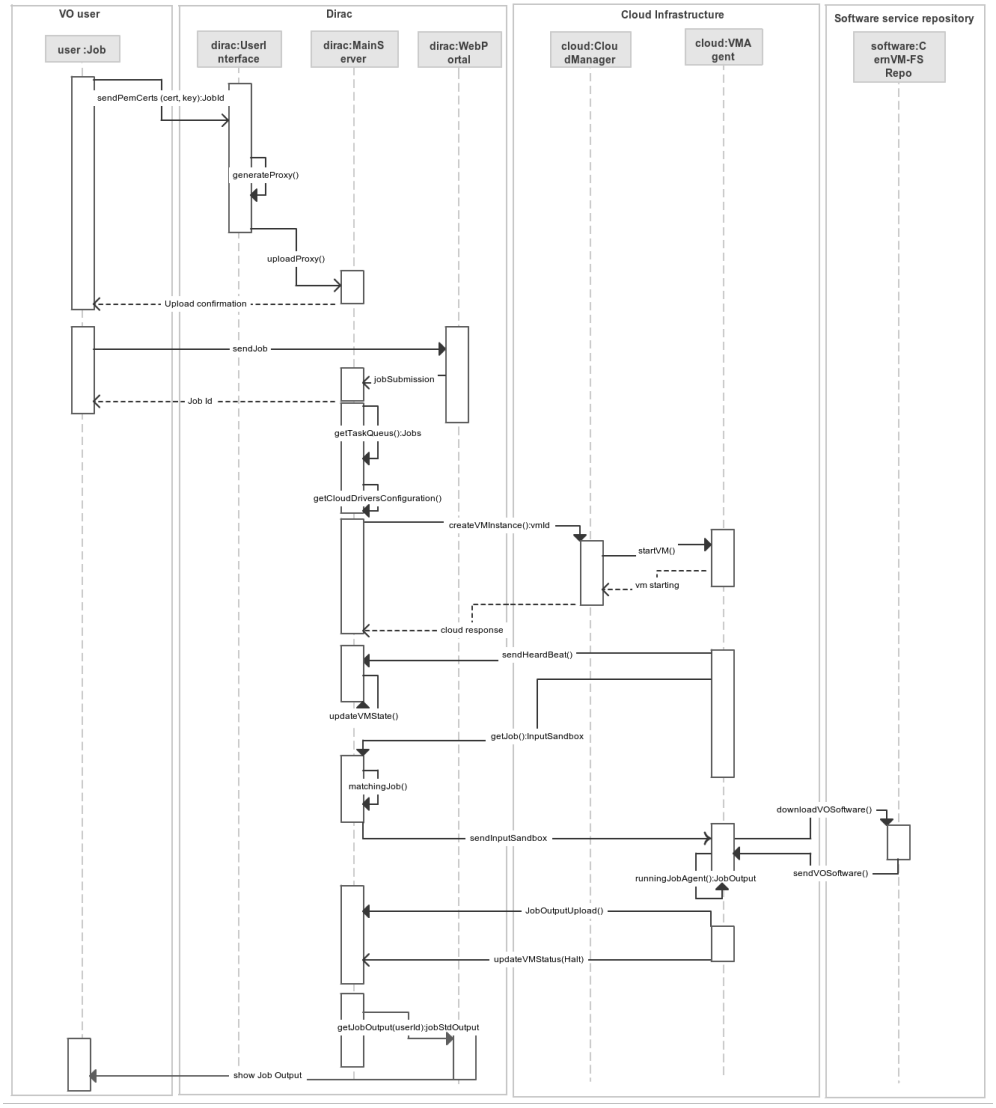


Figure 4.5: Sequence diagram of Multi-EndPoint prototype and Software Service repository.

The DIRAC integration with Multi-EndPoint is based in the previous prototype and it works in a similar way to the extension of DIRAC to use Amazon EC2 [57], which has been proven as a robust and extensible software to provide computing resources for the Belle scientific community on the Amazon EC2. As a learned lesson, Multi-EndPoint prototype can be used with different Cloud managers with several versions of them and job submission to different SITES.

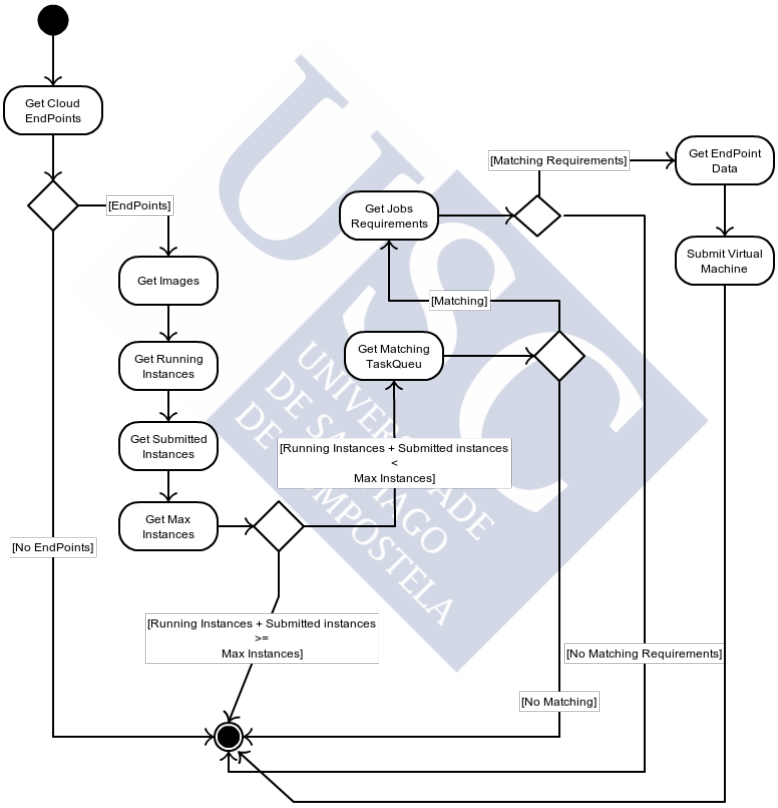


Figure 4.6: Activity diagram of the VM Scheduler component.

This second prototype gets the Cloud EndPoints, as is shown in Figure 4.6. EndPoints are the list of Cloud Managers that have been configured previously in the DIRAC CS [25]. The pending tasks are obtained from the central TaskQueues matching them with the capabilities

of the defined images. If not enough VMs are available and the maximum VMs threshold is not reached, then it submits a new VM using the specific VM Director. The VM Scheduler is in charge of submitting a new VM with DIRAC pre-installed and configured, which executes the Job Agent, such as is showed in the Sequence Diagram of Figure 4.6.

The DIRAC main server is in charge of the DIRAC Workload Management System (WMS). It acts as a middleware integration layer between different virtualization solutions. On one hand, it allows the interaction with other components, containing the logic flow of the VM scheduling engine. On the other hand, it is responsible of the persistency of the VMs information, through an associated database. Images and endpoints are described in the DIRAC Configuration.

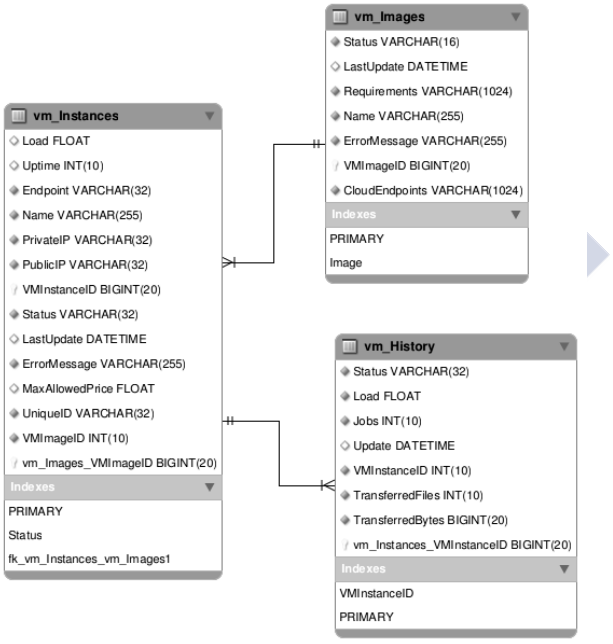


Figure 4.7: Entity–relationship diagram of MultiEnd-Point Prototype

The persistency of the information is another important part of the Multi-EndPoint prototype. The database stores the historic information of EndPoints or Cloud Managers that are linked with DIRAC, as well as the virtual machine instances that are stored in these End-

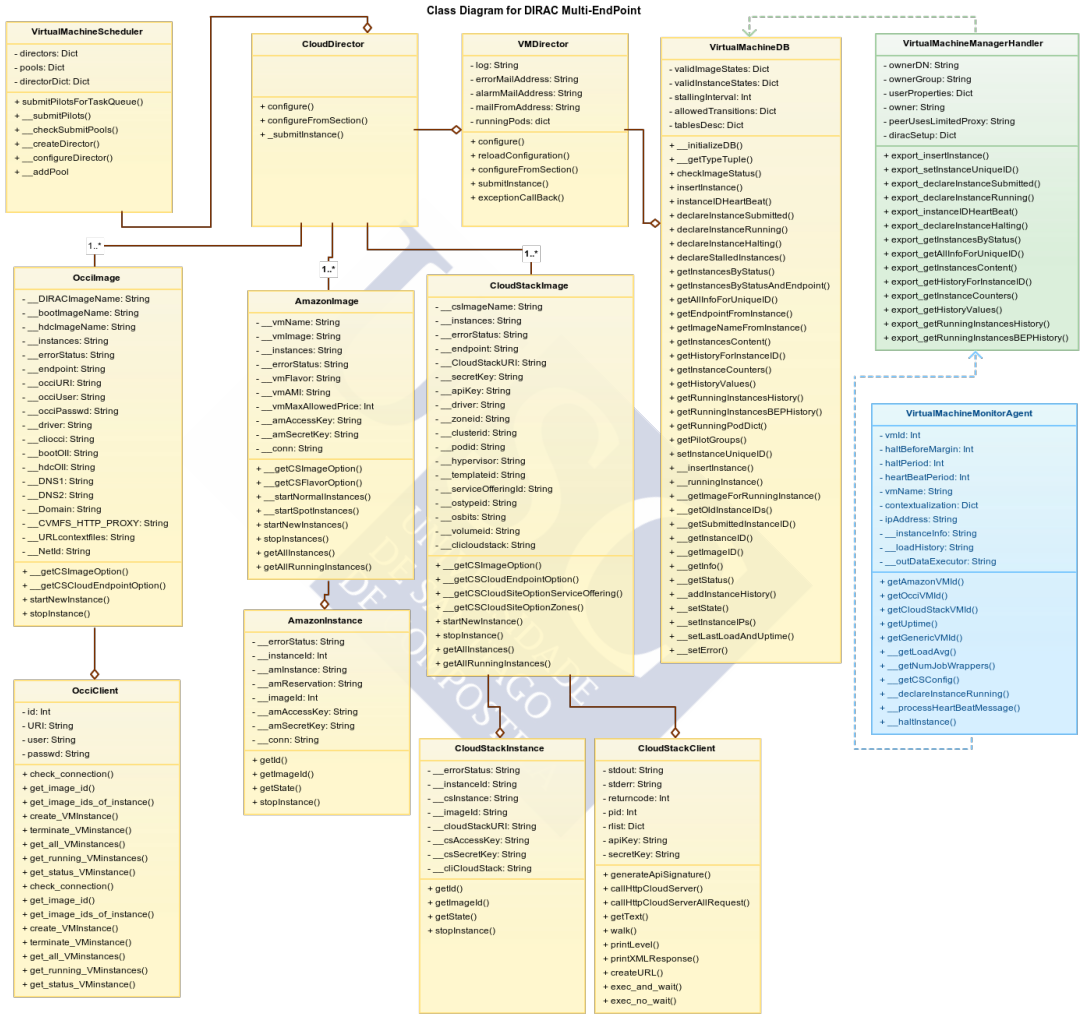


Figure 4.8: Class diagram of MultiEnd-Point Prototype, with the Virtual Machine Agent and Virtual Machine Manager Service

Points. The change of status of Virtual Machines is stored in the database too. Figure 4.7 shows the Entity-relationship diagram of the database developed in this thesis.

As a result of the infrastructure Sequence and Activity diagrams, the diagram of Figure 4.8 shows the Class diagram for the Multi-EndPoint Prototype. The yellow classes, with VirtualMachineScheduler class, define the variables and methods to interact with several Cloud Managers, as well as a Data Access Object (DAO) layer with methods that access and modify the database. The green class, VirtualMachineHandler, includes the variables and methods to manage the database externally. Finally the blue class, VirtualMachineMonitorAgent, is in charge of updating the virtual machine status.

The VM Monitor Agent is executed on the VM, starting immediately after the start up of the operating system. Its first action is to declare the VM running state to the VM Manager-Handler class.

The feedback provided by MultiEnd-Point prototype was used as the base to create the **Rafhyc (Resilient Architecture of Federated HYbrid Clouds)** [98] architecture, that is the final prototype of this thesis.

4.3.2 CloudStack prototype

This section describes the architecture design of the first prototype. This prototype integrates DIRAC and CloudStack, that will be part of the Formiga-Cloud project infrastructure. It is based in previous designs [57], which were adapted to CloudStack and the additional requirements. New components are required to perform this integration based in the requirement analyses and the architecture model. The main contribution of this thesis to the CloudStack prototype is the design, development and testing of the prototype. After the testing phase will be created user and admin documents about the infrastructure, and scripts that were be able to start automatically all the infrastructure.

In Figure 4.9, a component diagram of the integration of DIRAC and CloudStack is shown. The *VM Scheduler* agent starts working once it found at least one VM image that is defined in the configuration. This component gets the list of jobs to run from the central TaskQueues, matching pending tasks to the capabilities of the defined images. If there are not enough available VM, it checks if there are available nodes and, if so, then it makes a call to the Virtual Machine Node Tasks component in order to switch on more VM.

If there are available resources, it checks if there is enough CPU time required by pending tasks and if the maximum number of VM running limit is not reached through node tasks

component. If there is enough CPU time available in some of the virtual machines with running state and with some job agent component available, it submits the Job to this virtual machine.

The *VM Scheduler* takes the role of *DIRAC PilotDirector* used to interface DIRAC with Grid computing resources. Instead of submitting a pilot that installs DIRAC and executes a *Job Agent* on a WN, a full VM with DIRAC pre-installed and configured to execute the job and the *VM Monitor Agents* is submitted.

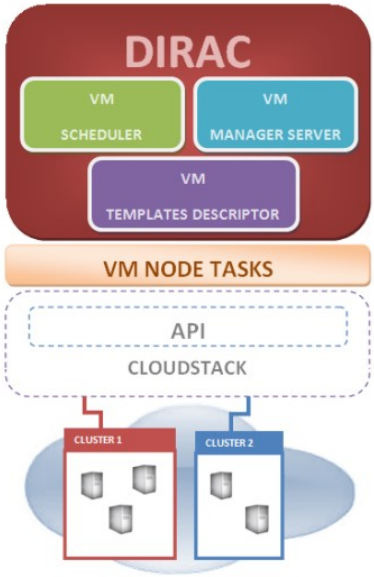


Figure 4.9: Component diagram of the integration of DIRAC and CloudStack

The *VM Manager Server* works as the middleware layer. It allows interacting with other components as Virtual Machine Image Descriptors, Virtual Machine Node Tasks or Virtual Machine Scheduler. Furthermore it contains the logic control of the application, and it receives requests in order to create actions such as the creation of new instances with some specific operating system or the start of a new virtual machine in an arbitrary node. On the other hand this component will take care of the persistency of the information relative to VMs. It has a associated database.

Before being used, the description of the new templates must be included in the DIRAC configuration, defining the following parameters.

1. **Specific CloudStack parameters**:
    a) *API key*: key to add to the sign in order to verify the SHA1-mac.
    b) *Secret key*: key to sign the previous message of the API, before to verify the SHA1mac.
    c) *Name*: name of the template.
    d) *Ostypeid*: the ID of the OS Type that best represents the OS of this template.
    e) *Requireshvm*: true if the template requires HVM, false otherwise.
    f) *Volumeid*: the ID of the disk volume the template is being created from.
    g) *Format*: the format for the template. Possible values include QCOW2, RAW, and VHD.
    h) *Hypervisor*: the target hypervisor for the template.
    i) *URL*: the URL of where the template is hosted.
    j) *Zoneid*: the ID zone of the template.
2. **Specific DIRAC parameters**:
    a) *UniqueID*: unique identifier of the Image as returned by the Cloud provider.
    b) *MinCPU*: minimum amount of cpu time required before a new VM request is made.
    c) *MinLoads*: minimum load required on the VM, when it drops below this threshold for a certain period of time the VM is halted.

The methods provided by the VM Manager will be: the creation and registration of new instance from a template in order to launch Virtual Machines; calls to the Cloud API; declare the submission of a VM while inserting the unique identifier of the VM instance; start a VM; stop a VM; and see the current VMs status. All the information arriving through these methods is kept in the Database. The VM Manager periodically checks for stalled VMs that have failed to report their activity through the heart beat mechanism for an extended period.

The VM Templates Descriptor component provides the logic application of the template creation. With some of the parameters described previously should be enough to create a new specific template in order to launch new specific VMs. It should interact with the VM Manager component to take care of the persistence of the information of the new created templates.

The VM Node Tasks component provides all necessary methods to make calls to the CloudStack API. It is an important part in this implementation of DIRAC, otherwise the DIRAC server wouldn't have any interaction and communication.

Finally, the Job Agent does the matching of user Tasks from the central TaskQueue and supervises the correct execution of the user Task on the Virtual Machine resource, reports periodically the activity and takes actions in case of abnormal behaviour or conditions. This component will connect to the DIRAC WMS matcher to request a new task pending for execution. All connections are secured using DISET mechanism and server certificate key-pair available in the VM and previously declared in the DIRAC configuration. This prototype results will be shown in the last chapter, and the results provides feedback for the second prototype or Multi-EndPoint Prototype.

4.3.3 Multi-EndPoint prototype

The Multi-EndPoint prototype section describes the DIRAC integration with Cloud managers. The multi-VO and multi-platform job submission is based in a bottom-up architecture. Starts with the low level DIRAC modules or subsystems, and progresses upward to the design of the Cloud managers, with their main modules, or main subsystem. To determine the order of execution, a structure chart is needed, and, to complete the bottom up design, the development of drivers is needed for each of the Cloud managers. In this way, bottom up design makes it easy to reuse code blocks. This part is based in the previous prototype, which has been proven a robust and extensible software to provide computing resources for Aula Cesga with around twenty computers (Formiga-Cloud project) and in the LHCb TIER-2 of the Universidade de Santiago de Compostela [129]. In this section, the main components of the design and implementation are described, providing the basic Virtual Machine functionalities to design and improve the integration of DIRAC, CloudStack and the OCCI/OpenNebula. The main contribution of this thesis to the second prototype is the integration of CloudStack in the Multi-EndPoint design, the test of the prototype and analysis of the results.

There are two servers and two VM components. Their mutual relations and their interactions with other DIRAC components are shown in Figure 4.10 and Figure 4.11. In Figure 4.10 the services/servers items have been coloured in light blue, the client and interfaces in green and red, the DIRAC agents in blue, and in Figure 4.10 the VM components have been coloured in yellow, the DIRAC server in red and the Cloud manager in blue.

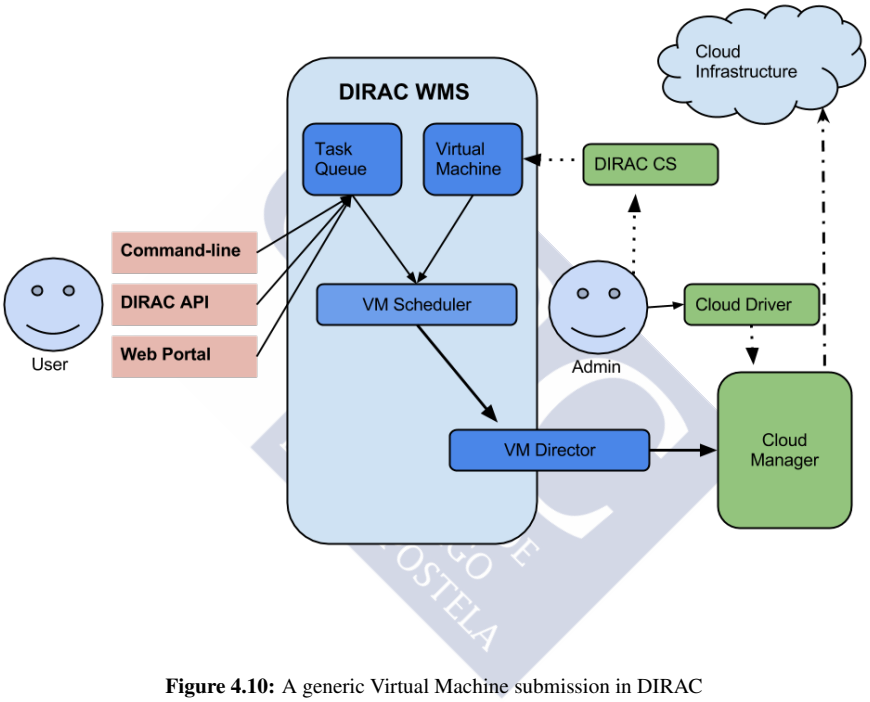


Figure 4.10: A generic Virtual Machine submission in DIRAC

In this prototype the VM Scheduler component gets the list of tasks to be executed from the central Task Queues, by matching the pending tasks to the capabilities of the defined images. If not enough VMs are available and the maximum VMs threshold is not reached, then it submits a new VM using the specific VM Director.

Figure 4.10 shows the general interaction of the VM Scheduler with other components to submit a VM to a generic Cloud Manager, which can be CloudStack, Amazon EC2 or an OCCI compliant, i.e. OpenNebula. The DIRAC Admin has previously uploaded the image to the Cloud Manager using the corresponding Cloud Driver, and set these values on the DIRAC Configuration.

The Virtual Machine Scheduler starts a full VM with DIRAC pre-installed and configured to execute the Job Agent, together with a VM Monitor Agent.

As in the case of the first prototype, the Virtual Machine Manager Service is the part of the DIRAC Workload Management System (WMS). It acts as a middleware integration layer between different virtualization solutions. On one hand, it allows the interaction with other components. Furthermore, it contains the logic of the VM scheduling engine. On the other hand, this component is responsible for the persistence of the VMs information, through an associated database. Images are described in the DIRAC Configuration. For these Images, the following parameters must be assigned:

1. **MinCPU**, is the minimum amount of CPU time required to trigger the instantiation of a new VM.
2. **MinLoad**, is the minimum load required on the VM. This parameter is read by the VM Monitor process running inside the instantiated VM and, when the load drops below this threshold for a certain period of time, the VM is halted.
3. **Flavor**, is the resource provider for which the Image is valid, i.e., Amazon. The same image can be reused under multiple flavors and/or Cloud providers.
4. **MaxVMs**, is the maximum number of VMs allowed to run simultaneously in a given resource provider.
5. **Requirements**, is the dictionary defining the computing capabilities of the VMs, used to match tasks.

As in the case of the first prototype, the VM Monitor Agent executes on the VM, starting immediately after the start up of the operating system. Its first action is to declare the VM running state in the VM Manager component. If there is an error in the connection or if any abnormal condition is reported back by the Manager, the VM is halted. The VM Monitor periodically monitors the CPU load of the VM, the number of executed tasks and the amount of output data of the VM. This information is reported to the *VM Manager* in heartbeats.

Finally, if the VM Monitor detects that the load of the VM has dropped below the configurable threshold, the VM is halted by the VM Manager.

The Virtual Machine Job Agent does the matching of the tasks from the central TaskQueue and supervises their correct execution on the VM resource. It reports periodically the activity and task actions in case of abnormal behaviour conditions.

This component connects to the DIRAC WMS matcher to request a new task pending for execution. All the connections are secured using the DISET mechanism and server certificate key-pair available in the VM, previously declared in the DIRAC configuration as in the first prototype.

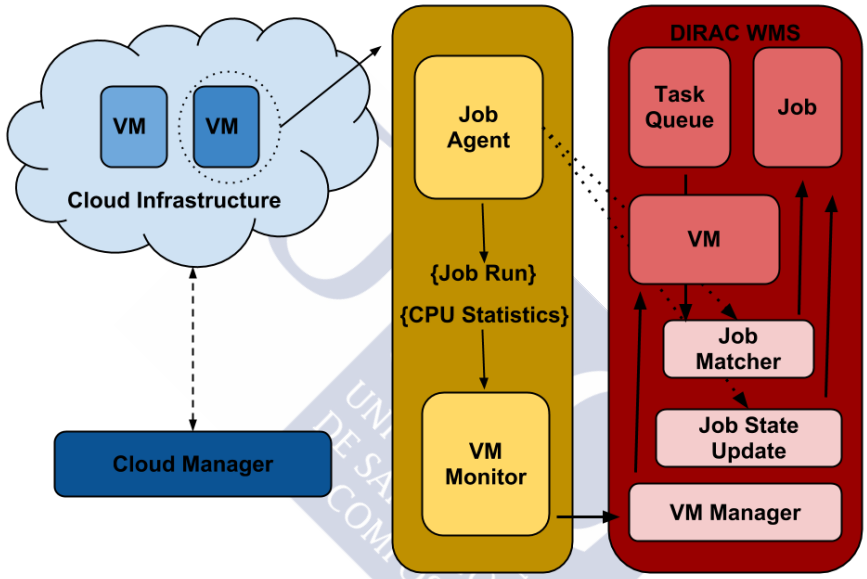


Figure 4.11: A generic Virtual Machine running Job in DIRAC

As is showing in Figure 4.11 the Job Agent and the VM Monitor are interacting with the DIRAC WMS. The communication between the two agents on the VM is indirect. The VM Monitor gets system statistics to learn about different states of the Job such as stalled or finished and it reports to the VM Manager. On the other hand, the Job Agent is interacting with the WMS for the Job matching and manages different states such as downloading input sandbox, really running or job completed. This functionality of the Job Agent is the same, independently of whether it is running in a VM or in a standard job in a real Worker Node.

If the VM Monitor detects that the load of the VM has dropped below a certain configurable threshold, the VM is halted by the VM Manager Service if there are no pending transfers ongoing.

The Multi-EndPoint prototype was built using **CernVM** with the OpenNebula Cloud manager. CernVM is a Virtual Software Appliance designed to provide a complete and portable environment to develop and run LHC data analysis on any end user computer and batch node. It contains only a minimal operating system required to bootstrap and provide access to the experiment software. This experiment software is delivered from CERN repository using the CernVM-FS that decouples the operating system and the software area. CernVM allows contextualization using the HEPIX standard [22]. The contextualization is used to make DIRAC available in the VM and to start the necessary components described previously.

The CernVM instances are automatically managed by the VMDIRAC OCCI development [132], including creation, status report, and halting. The VMDIRAC integration with OCCI/OpenNebula is dealing with the following images:

1. A CernVM batch node. The CernVM image specifically designed for Grid and Cloud Computing.

2. On the fly configuration image. On the submission of a new VM, an HEPIX Context section is specified to provide the environment variables. VMID (the current VM ID) and necessary variables for the CernVM-FS client and the network configuration. The OCCI Cloud manager creates a temporary ISO image with this environmental variables, and HEPIX booting system includes them in the system.

3. An ISO context image. It is used by the CernVM HEPIX compliant booting system to configure the contextualization. First, it obtains the DIRAC VM service certificate and key for the DIRAC VM agents. Second it triggers the CernVM-FS client repository configuration and the network configuration. This configuration is done before the network services are started (prolog.sh script in the ISO). The last booting init process of the CernVM HEPIX (epilog.sh script in the ISO) is in charge of configuring and starting the DIRAC Virtual Machine Agents.

With the feedback provided by Multi-EndPoint prototype the next section covers the a new Federated Hybrid Clouds Architecture design.

4.3.4 Federated hybrid clouds architecture

As the third prototype of this thesis, and based on the previous prototypes, a federated hybrid Cloud was designed and tested with a MANOVA Analysis. This architecture is named **Rafhyc**

(for Resilient Architecture of Federated HYbrid Clouds). The main contribution of this thesis to the Rafhyc prototype is the validation with CloudStack and analysis of the results.

Figure 4.12 shows an overall architecture of Rafhyc. The bottom of Rafhyc has two low level layers, to deal with the resources and the additional federated community services. The main layer is supporting a single point of access. The top of Rafhyc is using two interfaces to allow the integration of science gateways, pilot factories, job factories, science workflows or scientific applications. The Rafhyc Service Architecture is designed to construct resilient applications on federated Clouds. The architecture is composed of Layers, each of them, providing an overall functionality. Furthermore, a Layer is divided in Components for the different functionalities of the Layer. There are two low-level layers: Rafhyc Multi-Cloud Layer and Rafhyc Federated Hybrid Cloud Services Layer, supporting the resource access abstraction. The high-level layer, Rafhyc Resilient Services Layer, provides the resilience of the service delivery.

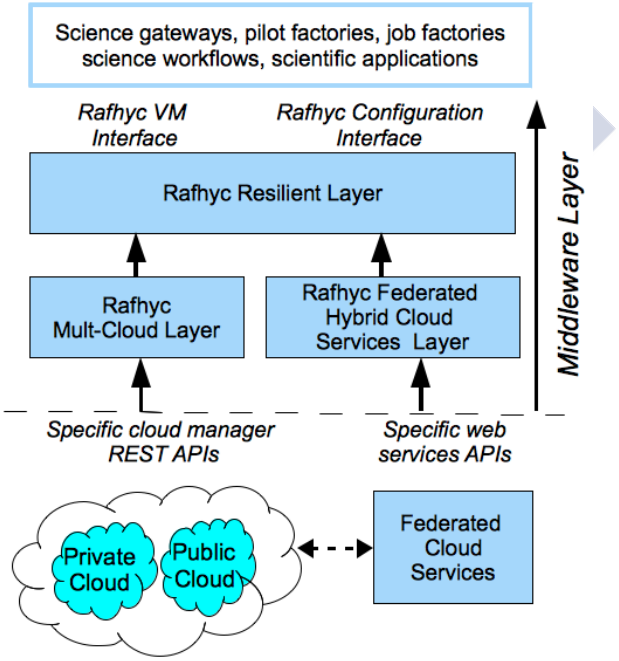


Figure 4.12: Rafhyc Overall Architecture.

The Rafhyc architecture defines a Layer breakdown in Components, following a bottom-up scheme:

1. **Rafhyc Multi-Cloud Layer**: low-level layer to access private and public end-points. Components:
   a) The open standard OCCI driver and the industrial standard EC2 driver, or any specific Cloud manager driver.
   b) Image Context Manager to deal with the different policies [21] of the resource providers.
2. **Rafhyc Federated Hybrid Cloud Services Layer**: low-level layer to access the Federated Services interfaces. Components:
   a) At least the drivers to access the Information and Monitoring Systems.
   b) Federated Services Polling to unify the access to different federated services.
3. **Rafhyc Resilient Services Layer**: high-level layer to provide service delivery for scientific applications. Components:
   a) Persistent Configuration
   b) Cloud and Federated Authentication
   c) Configuration Interface
   d) VM Launcher
   e) VM Status Manager
   f) VM Interface
   g) Availability Matcher
   h) Efficiency and Price Optimizer
   i) Benchmark

Persistent information is composed by configuration values that are not expected to change with status of the federated resources. Rafhyc also defines dynamic information, which can change with the availability and status of the federated resources. Dynamic information is updated by the Federated Services Polling and it is necessary to adapt to the dynamic and

unpredictable nature of the federated hybrid Cloud model. Moreover, Rafhyc components can be described attending to their behavior. Services are attending asynchronous request and triggering actions. Clients, Drivers and Interfaces support different functionalities for the Services. Clients are internal Rafhyc utilities, while Drivers and Interfaces are binding the bottom and the top of the architecture. On the bottom, Drivers integrate the necessary functionalities of the federated service APIs and the Cloud end-point APIs. On the top, Interfaces are supporting the binding with any application adopting Rafhyc.

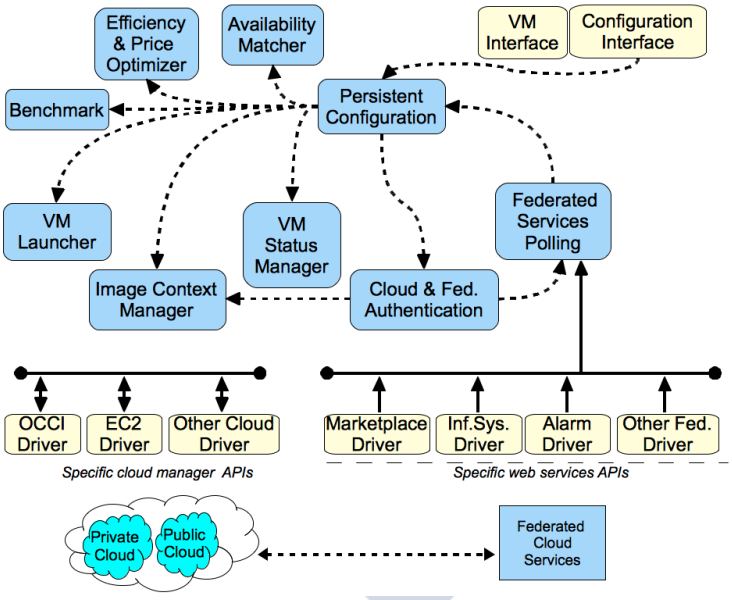


Figure 4.13: Rafhyc Persistent Information Flowchart.

Figure 4.13 shows the Rafhyc flowchart of the persistent information. Service components are in blue (b/w in gray), Client, Drivers and Interfaces are in yellow (b/w in very light gray). The doted lines represent information flows of persistent information. The Rafhyc Service is configured using the Configuration Interfaces and the information collected from the Federated Cloud Services.

Figure 4.14 shows the corresponding flowchart for dynamic information. On one hand there is the information polled from federated services, like the image metadata or the Monitoring notification by the Alarm Driver. Other examples of such information are the availability of the end-poins, the image availability on different end-points, the number of free VM

slots, or CPU or Credit available for a particular user. On the other hand there is the information generated by some components of the Rafhyc, stored as statistical records, such as the VM Status Management giving the VM failure rate for each image and each end-point, or the Benchmark records of computing efficiency.

Both Figure 4.13 and Figure 4.14 show two dimensions of the information flow between the components as a mix of persistent and dynamic information, depending on the nature of the origin of this information: configuration values (persistent) or system components status (dynamic).

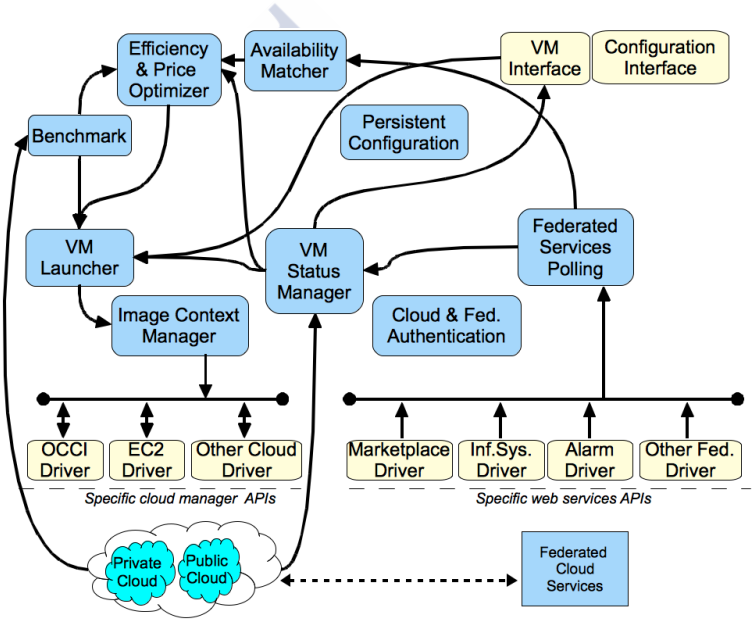


Figure 4.14: Rafhyc Dynamic Information Flowchart.

Description of Rafhyc components

This subsection describes the components of the persistent and dynamic flowcharts on Figure 4.13 and Figure 4.14, attending to what they are doing and how they are obtaining the necessary information.

Persistent Configuration

The configuration is included in Rafhyc in two ways. The core configuration is managed through the Configuration Interface, by manual or automated operations integrated in a high lever application. Starting from this core configuration the Federated Services Polling is able to discover additional configuration information, such as image metadata from a Federated Marketplace. Furthermore, the Persistent Configuration allows to define the User and Tenant information.

Cloud and Federated Authentication and Authorization

This component is in charge of Cloud Authentication methods associated to different user access to the end-points, and some required Federation Authentication with administrator role to services like image metadata catalog. The Authentication is dealing with different methods depending on the resource and the IaaS providers. In general terms the authorization to use the resources would be a third-party system. As a particular implementation example, EGI Federated Cloud is considering user / password for testing purposes and it is targeting a production level using X509 proxy with VOMS attribute to match a user to a specific EGI Virtual Organization (VO). In addition to the end-points, some community services, like image metadata catalog, are supporting token Authentication, but this is transparent for Rafhyc, a front-end of web server proxy is doing the token management.

Federated Services Polling

The Federated Services Polling is attending two types of requests to poll the Federated Services. One type is on demand request from the Services. The other type, triggered by asynchronous alarms notifying a federated Cloud update, polls the corresponding Federated Services and updates the Persistent Configuration. VM Status Manager Rafhyc tracks the VM status for statistical and monitoring purposes. Rafhyc images have to include remote notification of the status update to the VM Status Manager, on the starting of the VM running, on the VM halting and a periodical heartbeat notification to identified stalled VMs. Rafhyc VM status are submitted, running, stalled and halted. Thus, VM creation overhead is the time gap between the submission of the VM creation request and the VM running. Furthermore, the failures are recorded in the Virtual Machine database and the failure rate of an specific Cloud end-point can be obtained for a given period of time. The failure rate obtained for a time window close to the submission is used as a metric of the reliability of a Cloud SITE. In

the same manner, the VM creation overhead of a time window on a Cloud end-point, would be a metric to obtain the computing efficiency.

VM Launcher

The VM Launcher sends a request of VM creation to a specific Cloud end-point, based in the matching and optimization chain. Those services are using historical statistics as explained below. In this manner, the resilience problem is an exogenous parameter of this component. The VM Launcher is creating on-demand VMs as requested by the application adopting Rafhyc, through the VM Interface. VM Launcher is also submitting the Benchmarking VMs, which are direct submissions without the matching and optimization chain.

Image Context Manager

This component is in charge of the image contextualization. It is supporting specific scientific computing with ad-hoc images and context images. The contextualized image management is based in a practical approach to virtualization in HEP [77]. For this purpose, the Persistent Configuration has at least one Running Pod, which defines the relationship between a Rafhyc image with a list of end-points. A Rafhyc image contains a Boot image and the necessary configuration for a particular Contextualization method. This schema allows the use of golden images to simplify the image management. Instead of a golden image depending on a platform [21], a generic VM image can be configured using a ssh contextualization, if an in-bound connectivity is available in the VM for ssh and sftp operations. The context is a common setup of the VM to the PaaS or SaaS supported by Rafhyc. Moreover, a specific context setup of a particular end-point is included in the VM submission.

Availability Matcher

The Persistent Configuration includes a list of the end-points supporting a particular image. This information can be obtained from the image metadata catalog. An image may or may not be uploaded in a particular end-point, the Availability Matcher returns which end-points have the image upload and if such end-point is available to submit a VM. A Marketplace can be used for image metadata and also can be in charge of broadcasting of the images to all the corresponding Cloud end-points, but this is not always the case. For example, the Marketplace in the EGI Federated Cloud does not have such functionality by March 2013. Additionally, an image may be uploaded at a specific end-point, but the user may have no more slots for simultaneous VM running in the end-point. This resource constrain is not contemplated in the common elastic Cloud philosophy, but it has been introduced in the federated hybrid Cloud model coming from the Grid environment. Thus, Rafhyc keeps track record of

how many VM slots are given for a user at each SITE or end-point. Furthermore, the federated hybrid Cloud information system can publish dynamically the overall availability of the resources.

All this information is collected by the Availability Matcher to select a set of Cloud end-points available to submit the VM creation of an image. In this manner, Rafhyc is avoiding submissions to Cloud end-points that are going to queue the request until resources are free. The Availability Matcher is saving user response time and optimizing the virtual resource distribution because it is sending the VM to the Cloud end-points available for running.

Benchmark

Benchmark is used to measure the efficiency of the different end-points VM running slots. This component is periodically submitting a set of benchmarks to each Cloud end-point. A service listening gets the response with the metrics form the VM benchmark. The multiple Cloud providers have different efficiency in the compute running. This efficiency may change, so the benchmarking has to be proactive, but it does not need to be as frequent as real VM creations. Rafhyc benchmarks may include a by default workload type that would be used for a generic VM creation request. Additionally, different benchmarks would be defined for intensive computing or high IO computing. The information is recorded and the most recent benchmarks are used to obtain the computing efficiency, together with the SITE reliability and the VM creation overhead.

Efficiency and Price Optimizer

This is a central service for the resilience of Rafhyc. As it is shown in Figure 4.14, the Efficiency and Price Optimizer is receiving inputs from Availability Matcher, Benchmark and VM Polling. The Availability Matcher providers the list of available Cloud end-points to create a VM from a particular image y. The set of possible solutions is $S = e_1, e_2 ... e_m$ for a given image y, where y is not available to VMs creation in other end-points in this specific time. Then, the Efficiency and Pricing Optimizer takes advantage of the VM Status Management statistics to obtain the current VM computing cost of each end-point for i=e_1 to i=e_m, i $\in$ S, as it is shown in (1) and (2). The equation analyzes certain time window of VM submission to each end-point. The first fraction obtains the average of the runnable uptime, value 1 would be a theoretical VM without overhead and value 0 would be a VM without runnable uptime. This fraction is multiplied by the inverse of the benchmarking of the end-point, to obtain the cost of the average runnable uptime to process the benchmark. So, the submission cost to the End-point can be expressed as:

$$C_i = \left(1 - \left(\sum_{k=1}^{n} O_{(k,i)} / \sum_{k=1}^{n} T_{(k,i)}\right)\right) 1/B_i \qquad (1)$$

Where $O_{(k,i)}$ is the VM creation Overhead described above, for the last k VM created in the end-point i. $T_{(k,i)}$ is the life-cycle time, including the VM status submitted, running and halted, for the last k VM submitted to the end-point i. B_i is the normalized benchmark associated to the end-point i and to the type of job of the VM to be submitted. The Efficiency Optimizer is also considering (2), the evaluation function between any i in S, and the best evaluated option until such evaluation time. And the heuristic evaluation function can be expressed as:

$$E_i = ((1 - f_i)C_i) + (f_i C_b) \qquad (2)$$

Where f_i is the failure rate on the end-point i for a certain time window close to the current time. C_i is the submission cost in (1) of the the current evaluated end-point i, while C_b is the submission cost of the best evaluated option up to the current evaluation. The E_i is balancing the cost of probability of success on VM submission to the i end-point with, in case of failure, the extra cost to send it to the best end-point option b. The evaluation function can be used with different optimization algorithms, a simple economical approach can use the evaluation function to obtain the optimizer end-point destination for a particular VM image in a specific time. Alternatively, the evaluation function can also support other optimization approaches such as natural inspired algorithms, genetic algorithms or graph path finding. Besides of the computing efficiency, this component is also doing a Price optimization. Nowadays, the economics philosophy of the hybrid Clouds is to use the pay-per-use Clouds when there are not available Community Cloud resources. For this purpose, two metrics are important, on one hand the Persistent Configuration has the price list of the commercial Cloud providers. On the other hand, the previous Efficiency Optimizer allows to normalize such prices in terms of computing efficiency in the specific time of the VM creation request, as it is shown in (3).

$$p_i = f_c(p_c)E_i \qquad (3)$$

In this formula p_c is the c price on pay-per-use, f_c is the function to normalize the c provider price policy into a common time unit for all the providers. E_i is the heuristic function associated to the pay-per-use end-point i, in the same manner as described above for Community Cloud end-points. The Price Optimizer checks all the possible options, and takes the pay-per-use end-point with lower normalized price to submitted a VM creation request. The QoS is considering not a theoretical provider SLA, but with the common benchmarking and reliability measures of every provide, which is within the heuristic function.

4.3.5 Big Data architecture

The use of Big Data endpoints over DIRAC interware had not been done before. Therefore, the first step was to design the appropriate architecture with the specific database and components. These new structures are described in some detail in the next subsections.

DIRAC and the interface for Big Data software

Currently, DIRAC has been using Grid resources with Pilot Scheduler Agent, together with middleware specific dispatcher (called PilotDirector) to populate Grid WNs with DIRAC pilot jobs. In the case of DIRAC and Cloud resources, the idea is to make use of VMs instead of pilots as container for the Job Agent. In the case of Big Data Software the use of Pilots was considered at the beginning of the project and implemented, but since the endpoint is going to be the SITE Big Data scheduler resource, which is in charge of managing the queues and make the Map Reduce tasks, this concept was moved to the direct Job Submission. The direct job submission is the basic idea for sending jobs to Big Data endpoints with a specific dataset. In the integration of DIRAC and Big Data Software, three main functional pieces have been identified, apart from other modifications to deal with this new integration.

BigDataDIRAC components

Three new components, to be executed on the DIRAC main server, were created to provide the necessary Big Data connectivity: the BigDataScheduler agent, which requests the execution of new Big Data jobs to the defined endpoints as necessary; the BigDataMonitoring agent, which controls the storing of the different job Status into a Big Data database (BigDataDB), reporting them to the JobDB, reporting the condition error (in the case of error), and moving, when the job finishes, the Job Output data to the OutputSandBox store; and, finally, the BigDataDirector modules, which allows the use of different Big Data software.

The process of job or pilot submission is showed in Figure 4.15. The first step of this process is the user job submission, through the DIRAC Wep Portal, to the WMS. The job is kept in the TasQueue, where there is a list of organized pending tasks. Each TaskQueue contains tasks with identical execution requirements, e.g., CPU time requested by the user, the required platforms to execute, or the identity of the submitter. These requirements are matched with the appropriate computing resources by dedicated DIRAC agents, that in the case of Big Data are called by the Big Data Scheduler Agent. Once are matched the pending

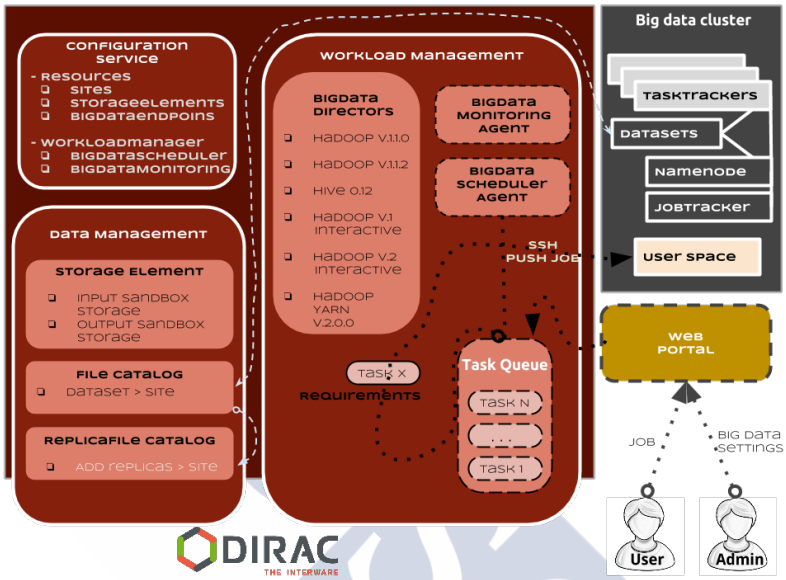


Figure 4.15: Diagram of the Big Data software and DIRAC integration architecture.

tasks with the capabilities of the Big Data endpoints, the BigDataScheduler Agent checks if the limitation in the endpoint of job number is reached. And with a very simple logic, if there is enough cpu time required by pending tasks, the maximum number of running jobs is not reached, and the Big Data settings and dataset match with the previous SITES defined with DIRAC CS, DIRAC FileCatalog and ReplicaCatalog, the job is submitted to the specific Director. This logic can be more elaborate if necessary. For instance, if several providers are defined that make available different hardware, the most efficient choice is to be found. For the purpose of the present prototype, the choice has been reduced to a single option, but a global scheduling optimization by load balancing will be explained in detail in following section.

The BigDataDirector module has been coded to submit the requests to different Hadoop software. For other resource providers, analogous BigDataDirector modules can be implemented. This component takes the role of DIRAC Job Director or Pilot Director [27] used to interface DIRAC with Grid computing resources. Instead of submitting a pilot that installs DIRAC and executes a Job Agent on a WN, the normal job is submitted directly to the JobTracker through SSH and the Hadoop Job id is obtained and stored in the BigDataDB. With

the Hadoop Job id the BigDataMonitoring Agent starts to inquire about the status of the job. The Monitoring request time to the Job Tracker depends on a polling cycle defined by the DIRAC admin. These agents are using the BigDataDirector for the creation of objects with information collected from the CS in the Resources and WorkloadManagement system sections. Finally, when the job has finished, the BigDataMonitoring gets the Output data, and depending on the configuration, it cleans or keeps the temporary folder and files. This agent also uploads the data to the Output Sandbox storage.

Submission methods

BigDataDIRAC has integrated three different methods of job submission, two corresponding to interactive and batch Big Data Job Submission and a third one to make use of High level languages such us Hive, of Pig. These submitters are being integrated with the BigDataDirector and they are explained in the following subsections in detail.

Batch Job Submission

The integration of different Big Data software is made possible by developing the specific interface for each Big Data software version. The Big Data Director is then made aware of the existence, via the CS, of this new software version.

The current versions developed for BigDataDIRAC are ranged from Hadoop v.1.0.0, Hadoop v.1.1.2 and Hadoop Yarn 2.0.0. All the methods defined in this classes are creating the SSH communication with Hadoop, with all the necessary requests to Hadoop Clusters. The sequence diagram shown in Figure 4.16 explains all the process with a Hadoop v.1.0. class, from the Job Submission in the BigDataScheduler Agent until the download of the job output data from the Hadoop Cluster with the BigDataMonitoring Agent. In the first part a XML wrapper is created, this XML wrapper has the style of a job configuration file [114], following the process described in [73]. All the relevant information that was attached by the user when the job was submitted through the Web Portal, such us the DataSet or the OutputData, is included in the Job Wrapper. In the next step, the XML wrapper is added to the job payload which is a self-extracting file that will be executed in the Hadoop, and then this payload is submitted to the Hadoop Cluster. Once the Hadoop Job Id is obtained, it will be stored in the Big Data database. This Hadoop Id will be used by the BigDataMonitoring to inquire about the status of the job. Finally, when the job has been completed, the job output is retrieved from the Hadoop Cluster and stored in the DIRAC OutpuSandbox.

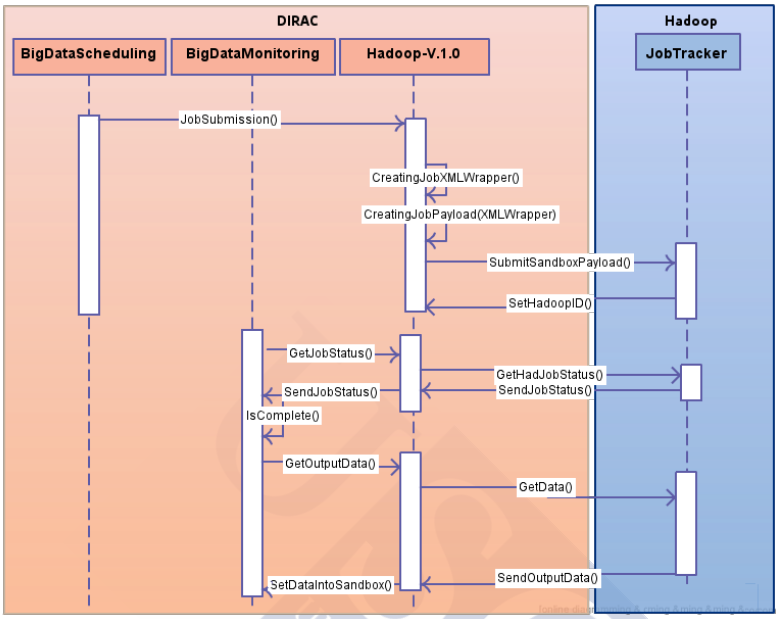


Figure 4.16: Sequence diagram of the Big Data job submission process.

Hadoop with interactive submission

The usual method to submit jobs to Hadoop is the direct job submission, and the interactive submission tries to emulate this process. Figure 4.17 shows the sequence diagram of one Hadoop job as interactive submission. In the diagram the red boxes represent DIRAC components and the blue one is showing the Hadoop part. First, the Hadoop interactive class creates the job payload with the typical Hadoop submit method, then the payload is submitted to the Hadoop cluster and the job output is stored in a predefined place, at the same time a thread with a monitoring system for catching the Hadoop job id is launched, it analyzes the job output and reports to the server which is the Hadoop id of the job. The monitoring process is similar to the normal job submission, but in the step of job status, when Hadoop interactive report to the monitoring that the current job is completed, it ask again to the Hadoop cluster if there is a new Hadoop job to be launched. This is because some Hadoop processes require of several Hadoop jobs to finish all their tasks. Finally, and in a similar way of Big Data job sub-

mission, if the Hadoop job is completed and the process does not require any additional jobs, the job output is obtained from the Hadoop Cluster and stored in the DIRAC OutpuSandbox.

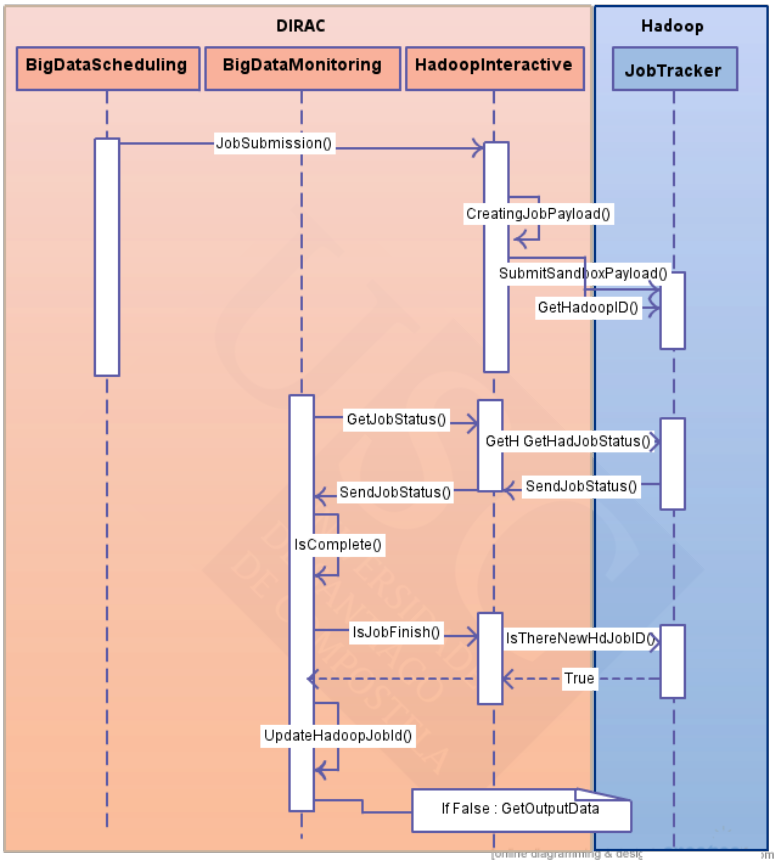


Figure 4.17: Sequence diagram of the Big Data job interactive submission process.

Job submission for High level languages with trapping system

BigDataDIRAC provides the SITES with the possibility of defining, under the utilization of Big Data Software, the use of High Level Languages such as Pig or Hive. In the case of Hive Director, the developing was based in Programming Hive book [24]. One of the main problems when developing the code for Hive is that it creates splitted hadoop jobs from the submitted job, and it makes difficult to control externally the entire process. Therefore in this

case we have created a trapping system which is launched from the Hive Director. Figure 4.18 shows how a Hive job is submitted and monitored. Initially, the BigDataScheduling Agent submits the Hive job and then the Director launches two threads to control the status of the job in all the transitions. The first thread launches the job and traps its output. The second thread monitors this job. There are three steps in the process of monitoring a job, the first one obtains the number of subjobs that will be created in Hadoop to execute the job, the second controls new jobs, and the last step controls the finalization of the jobs. When there are no more jobs, this monitoring last thread will get the job output and it will store it in the Sandbox of a DIRAC Storage.

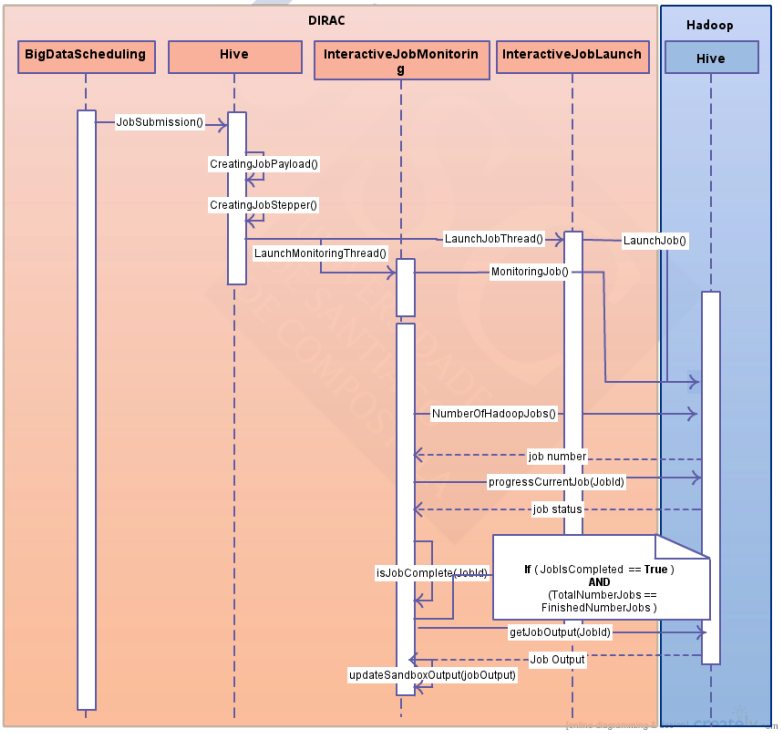


Figure 4.18: Sequence diagram of the Hive job submission process.

Big Data database and configuration

New settings need to be defined in the DIRAC CS for the new Big Data database. These settings are needed for DIRAC to know the available resources and how to interact with them. The database (DB) keeps all the necessary information to enable the communication between DIRAC and Big Data software. Before they are included in the database, the description of the new Big Data endpoints must be included in the DIRAC configuration by defining the following parameters:

1. SiteName: the SITE that stores the Big Data Cluster needs to be known by DIRAC, before defining the endpoint, the SITE and the Storage Element need to be added by the admin. These steps allow DIRAC to know where are the clusters, and the Storage Element and SITE vinculation since this information is important to include in the DIRAC File Catalog some specific dataset.

2. PublicIP: as the name implies this parameter contains the IP that allows DIRAC to communicate with the external world and, in particular, with the Big Data clusters. The SITE needs to provide an IP or some type of access to the computing resources.

3. User: the resources and HDFS access needs to be granted and assigned to an Hadoop user, this user needs to be specified to DIRAC in order to access the cluster. Different users could be defined for managing data with different privilege access in the same SITE, but then a distinct name should be used for each SITE and manager couple.

4. Port: the SSH or the task tracker port.

5. LimitQueueJobs: the DIRAC admin can define limitations on the number of jobs that will be submitted to a specific cluster.

6. Big Data Software and Version: DIRAC Directors need to know the software type and version in order to create the SITE Big Data object with the appropiate features required by the endpoint.

7. URL: the SITE could define the tracker endpoint in the case of having it.

8. UsePilot: the feature for Pilot submission could be enabled, but by default is disabled. The project architecture has been tested with direct job submission to the SITE.

9. IsInteractiveMode: this parameter allows DIRAC to catch and control the job as in the normal user interactive submission.

10. Requirements: dictionary defining the computing capabilities of the Cluster. It is used to match tasks to sites.

11. High Level Language: dictionary defining the high level language software (Hive, Pig, Spark etc...) and the version supported by the SITE.

These parameters are used by the DIRAC directors for the object creation, that allows the pre-configured endpoint match the requirements with the DIRAC TaskQueue, the submission of some Pilot or Job to the specific endpoint. The Big Data database keeps the Big Data job status with the endpoint information related with the SITE where the job was submitted. In the set of attributes kept in the main table of the database, the BdJobID is an auto increment primary key. This table also allows to relate this JobID with the DIRAC Job Id and the Big Data Job Id. Furthermore, the SITE Big Data cluster settings will be saved in this table too. The Scheduling and Monitoring agents will be in charge of updating the job status in the database. The DB also keeps the history of the Big Data jobs status and all the changes as reported through the Monitoring Agent, and depending of the pooling time defined by the DIRAC admin. The StalledJobAgent is in charge of checking for stalled jobs that have failed to report back their activity through the Monitoring mechanism for an extended period.

Others DIRAC databases are used by this setup, the FileCatalogDB, JobDB and AccountingDB are the databases that offer more utility in our setup.

Federated BigDataDIRAC workload management

The integration of geographically distributed Big Data clusters across different organizations presents a major workload management problem. This is a classical scheduling problem with several particular features, among the most important, it can be mentioned the heterogeneity of resources with different local existing scheduling policies and computing power capabilities, the large scale data systems with the possibility of multiple replication policies, the rapidly scaling requisites, the dynamic solution space or the multi-objective problem domain. This initial definition of federated Big Data scheduling is an NPC problem which requires to deal with heterogeneity, scale, complexity and dynamism.

Generalist aim could be expressed as an optimization problem. A large corpus in optimization theory usually identifies sub-problem heterogeneity to deal with two major components,

intensification and diversification. Intensification is focusing the solution search in a local region and should ensure the convergence to the optimal. On the other hand, at a global scale, diverse solutions require to explore the search scale to avoid local suboptimal. With these particularities, it would be interesting the stochastic heuristic processes instead of deterministic, because the problem can be described as multimodal (several best solutions) with no continuous distributions (because different nature of services and performances constraints). Turing was the first using the term heuristic search. In 1948 he described basics ideas of machine intelligence and learning, neuronal networks and evolutionary algorithms [125]. Other optimization heuristics have been developed and improved like support vector machine [130] and a wide range of supervised learning taxonomies, including hybrids [68]. Latter approaches introduce the use of a memory to adapt to dynamically changing problems, an example of such requirement would be the Big Data cluster readability or available power conditions changes. Thus, statistic history in the objective or constraint functions would be an advantage for service level predictions. Well known examples of such heuristics are tabu search [67] or nature inspired algorithms with ACO [41] and PSO [84]. The more recent nature inspired approaches like cuckoo search [140], or human inspired like harmony search [65].

On the other hand, generalist heuristics have some disadvantages compared with specific adaptations or simplifications. First is the theoretical limit of generalist optimizations for heterogeneous and complex environments. The recent work of [138] demonstrates that some algorithms outperform others for a given type of optimization problem, i.e. depending on the workload and running conditions there would be various problems and distinct algorithms will perform best. Second disadvantage is a high degree of software complexity in the migration of the theoretical algorithms to the real world, introducing important overheads, unexpected logic status and divergence in the software maintenance life cycle. These are not minor issues of the proposed work, since the aim it is not only a theoretical algorithmic approach, but also a real world solution.

Federated Big Data workload management with DIRAC has adopted a compromise approach, simplifying the problem in two levels of scheduling optimization: local and global. Local optimization is done at the Big Data clusters level and it includes vanilla Map Reduce scheduling provided by Hadoop, both V1 and YARN releases, but also high level Map Reduce ecosystems such as Pig, Hive or Spark. In this way, some of the scheduling issues are moved to the user application space, with particular requirements depending on the application. The intensification component of the optimization problem will remain at local

cluster and user application level, which is an approach to face the fact that some algorithms outperform others for a given type of optimization problem. On top of these possible local optimizations DIRAC integrates a global scheduling optimization. DIRAC Big Data scheduling is in charge of the diversification component to deal with heterogeneity. DIRAC job scheduling engine [27] has been enhanced for the particular federated Big Data features, covering job-cluster matchmaking problem by datasets and available technologies, DIRAC group job priorities, site availability and eventually other user application requirements. In this way; an NPC problem is simplified to a linear programming problem at global scheduling level, as detailed in the following subsection.

Global scheduling optimization by load balancing

Big Data federated and DIRAC have proposed an on-line model for global scheduling to be adapted well to the dynamic conditions of the distributed Big Data clusters. The diversification component of the optimization problems can be faced in a simple way by load balancing (LB). Other LB approaches in similar distributed computing environments have been proposed. LB between different cloud providers has been demostrated to save costs [122], which is an argument for the rationale of the presented approach. Both job completion and LB are targets of max-min strategy for job scheduling in cloud computing environments [45], including QoS constrains, in a similar way of DIRAC Big Data scheduling job requirements matchmaking. For example, to submit a job based in a particular ecosystem, to some of the clusters supporting such technology. Other per cluster QoS requirements could be defined at DIRAC user job level. More related work is found in algorithms for distributed computing LB, like the Hierarchical Load Balanced Algorithm (HLBA) for Grid environment [88] or the Dynamic Load Balancing Algorithm (DLBA) [117] or DLBA with genetic algorithms in [89]. These related works motivate the use of LB as a sensible approach to deal with the global scheduling in DIRAC federated Big Data.

DIRAC job matchmaking is taking into account the precise running conditions at the resource and last moment requests to the system. The DIRAC Workload Management System provides one single scheduling mechanism for jobs with very different profiles. To achieve an overall optimization, it organizes pending jobs in task queues. Task queues are created with jobs having similar requirements. A group job priority is assigned to each task queue. Pilot pre-allocation submission and subsequent pull job matching are based on these priorities, following a stochastic approach in each task queue and dynamic priorities in case of job rescheduling. Figure 4.19 shows the schema for Big Data scheduling in DIRAC, based in

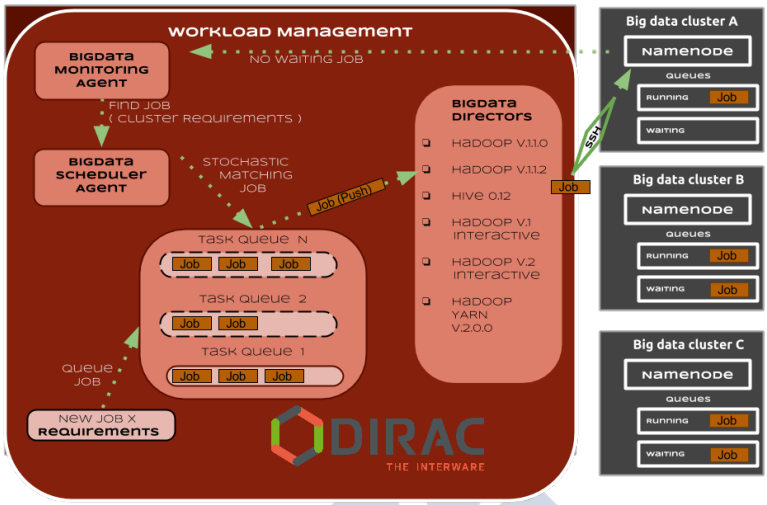


Figure 4.19: BigDataDIRAC scheduling system schema.

the Task Queue matching with available resources by a push job to the corresponding cluster. Vanilla local cluster schedulers with Hadoop are able to process jobs in sequence one by one. Actually, it is possible to do some job parallelization, in the case when some data nodes are free of map tasks, by only submitting Reduce tasks and starting an additional job to perform the map tasks in these data nodes when the Reduce task are done. For this reason, DIRAC Big Data scheduling is pushing with one waiting job to each cluster, whether there is pending workload in the tasks queues corresponding to those cluster resources, and maintaining the rest of the jobs in the central DIRAC tasks queues to latter pushing the job to the appropriated cluster. This dynamic matchmaking is able to provide system LB between distributed resources. LB aim is to optimize resource usage and a usual metric is the total makespan. At the same time, the general load conditions of the resource service will affect to the particular job completion times, so in an indirect way it is also expected a job response average time optimization. Section 5.4 of this thesis shows different use cases to test the optimization capabilities of the proposed workload management for federated Big Data. The main hypothesis is that the federated BigDataDIRAC workload management is able to take advantage of

the aggregated computing power of distributed Big Data clusters by an efficient global load balancing.

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

CHAPTER 5

EVALUATION OF THE PROTOTYPES

This chapter contains the validation process of the defined infrastructure. Tests and benchmarks were created in order to simulate real use-cases. The integration results with CloudStack prototype and software repository services, in the Formiga-Cloud project, are evaluated. The MultiEnd-Point prototype is validated under VMDIRAC development and federated hybrid Cloud prototype performances are evaluated. Finally, Big Data software and the integration with DIRAC is tested and validated.

5.1 Results and Evaluation of CloudStack Prototype (2011)

This prototype creates the infrastructure for the Formiga-Cloud project, and it is based in the design described in chapter 4. The final design of the **CloudStack Prototype** is shown in Figure 5.1 that defines the processes and services that are involved in the infrastructure.

The first step in the job submission is the registration of the normal user in the DIRAC CS. This registration is performed under the DIRAC admin role. DIRAC CS component provides specific setups for users, Cloud managers and DIRAC services, which are needed to run the job under a Virtual Organization. This process is shown in Figure 5.1. In a second step, the user has to choose one of the following 3 ways of job submission:

1. **The DIRAC API**: is a set of python classes designed for the user to easily access a large fraction of the DIRAC functionality. Using the API classes it is easy to write small scripts or applications to manage user jobs and data. The DIRAC API provides several advantages for the users, which are enumerated below:

a) Provides a transparent and secure way for user job submission.

b) Allows to debug locally the programs before being submitted.

c) Allows to run single applications or multiple steps of different applications.

d) The user can perform an analysis using understandable Python code.

e) Using local job submission the job executable maybe be executed locally in exactly the same way as it will run on the Worker Node. This allows to debug the job in a friendly local environment.

f) Using local submission mode the user can check the sanity of the job before submission.

g) All the DIRAC API commands may also be executed directly from the Python prompt.

2. **COMMAND-LINE in the User Interfaces machines**: this is the typical Grid submission, when the job to be submitted should be described in a JDL file.

3. **Web Job Launchpad**: this is the new way of job submission through the Web Page.

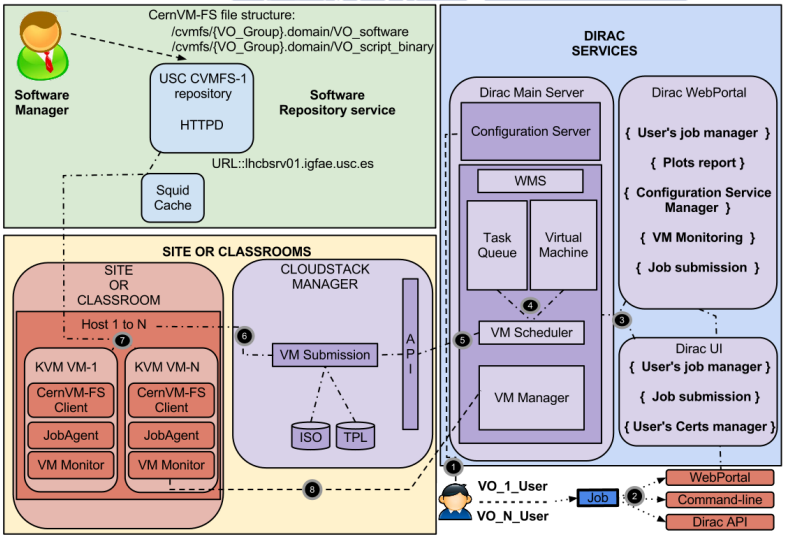


Figure 5.1: CloudStack prototype design in the Formiga-Cloud project.

After user submission, the job goes to the DIRAC Server, which traces the job in order select the best conditions to the VM submission. This step is done by the VM Scheduler component and it corresponds to the steps 3 and 4. In the step 5, the DIRAC Server sends the specific command to the CloudStack Server API [4], and the manager submits the VM, which is started in SITE hosts in the step 6. In this prototype, the SITE hosts are located in Aula Cesga as the Formiga-Cloud infrastructure. After the start up of the VM the CernVM-FS client connects to the CernVM-FS repository, which is hosted in USC TIER-2 and provides the software for the step 7. Finally, the VM-Manager that is running in the DIRAC Server gets a message notification from the Virtual Machine, confirming that the VM is in "Up status" and is running the user job in step 8.

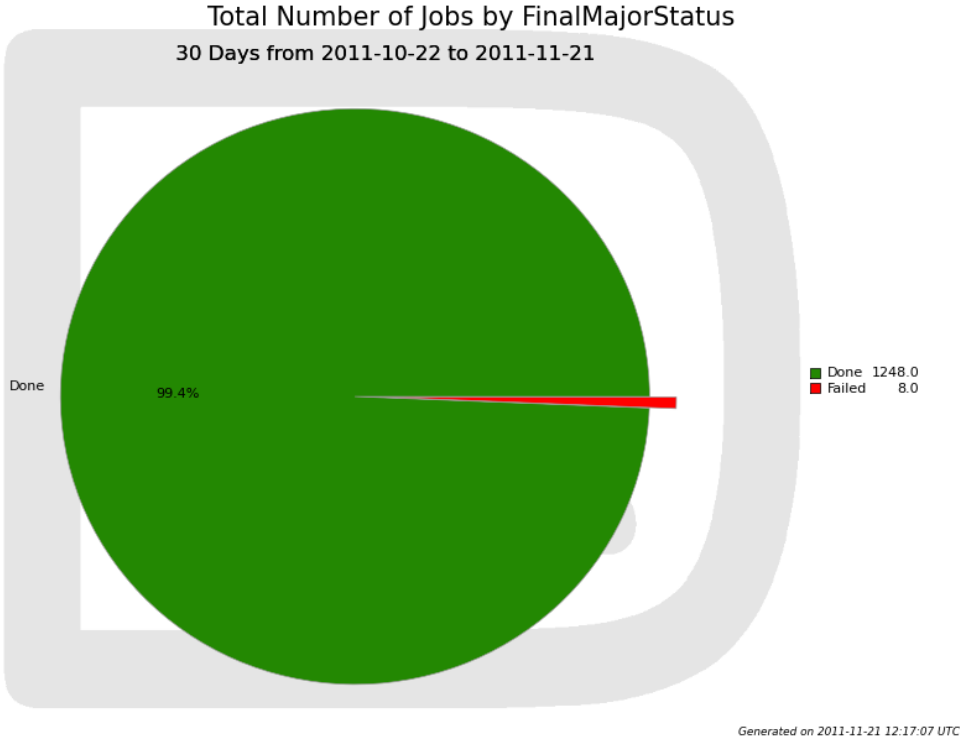


Figure 5.2: Total number of executed jobs in the CloudStack prototype.

Figure 5.2 shows the total number of jobs in the battery tests, which consisted in the submission of simple jobs via the DIRAC User Interface (UI) for several VO users. 99.4% of the jobs(1248) finished successfully.

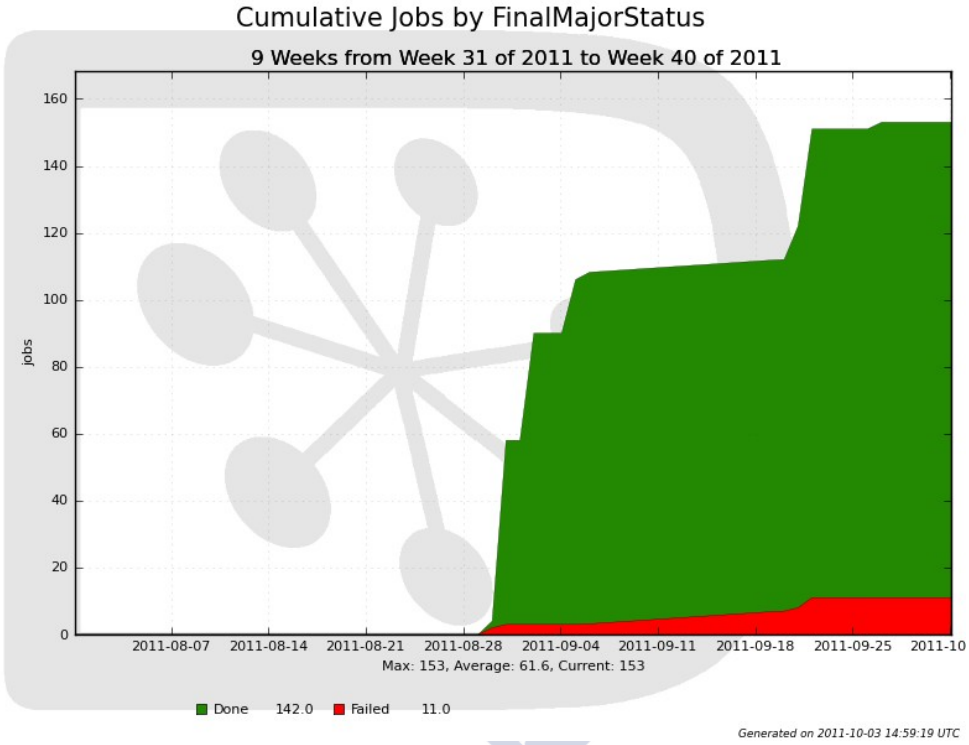


Figure 5.3: Cumulative jobs of DIRAC and CloudStack integration in Formiga-Cloud project.

Figure 5.3 shows the cumulative number of jobs. A small amount of failed jobs can be seen in red. The trace of these failures indicates they were due to some of the VMs were halted by the CloudStack manager. In the MultiEnd-Point prototype, those problems are solved by identifying the cause of the failures, and resubmitting the failed jobs. Furthermore, in this first prototype a new planner for CloudStack has been created to efficiently switch on VMs on non-dedicated resources. Benchmarks have been run with KVM over computer classrooms in order to identify the relevant information about the execution of VMs. The final goal is a better use of the computational resources available in the organizations.

5.2 Results and Evaluation of Multi-EndPoint Prototype (2012-2013)

A Multi-EndPoint prototype is created with the feedback provided by the CloudStack prototype and the DIRAC-EC2 integration. The main objective of this prototype is the integration of several Cloud managers in the VMDIRAC development, and also the creation of multi-VO and multi-platform job submission based in a bottom-up architecture.

5.2.1 Results for CloudStack with Multi-VO and Multi-Platform

This subsection describes the test results for Multi-VO and Multi-Platform [56], where the objectives were the proof-of-concept of the new Multi-EndPoint architecture with the CloudStack Cloud manager.

In the proof-of-concept process, simulations of several groups were executed in different platforms such as, Fedora 12, Centos 5.5 or Ubuntu 9.04, in order to test the Multi-Platform feature. The test infrastructure was using KVM hypervisor with 4 nodes, and each physical node was equipped with 2 quad-core processors IntelXeon X5355 @ 2.66GHz, and 16 GB of total memory. The jobs were classified in short jobs, with a time execution of 20 minutes, and long jobs, of around 8 hours of execution time. Figure 5.4 shows the infrastructure and the job submission steps. In this prototype there are several users belonging to different VOs. In the first step the users with DIRAC admin role makes a user list depending on the VO. Furthermore, in this first step, the DIRAC admin is in charge of adding the Cloud settings in the DIRAC CS, taking care of the different preconfigured images of the Cloud manager. The second and third steps are the job submission, and the job reception by the VM Scheduler component, from the TaskQueue.

Once that the job is in the VM Scheduler, specific Cloud information is obtained from the DIRAC CS according to the user credentials, this is the step 4. In this part of the prototype, each user has been configured in a specific VO, and each VO has been assigned to a unique EndPoint. The VM Scheduler component sends the specific EndPoint command to the CloudStack Server API [4], and the manager submits the specific image, which in this case corresponds to Ubuntu, Centos and Fedora, these are the steps 5 and 6. Then, the VM Scheduler that is running in the DIRAC gets a started message notification from the Virtual Machine, which confirms that the VM is in "Up status" and is running the user job in step 7. Finally, when the VM starts up process the CernVM-FS client connects to the USC CernVM-FS repository, which is hosted in USC TIER-2 and provides the software in the step 8.

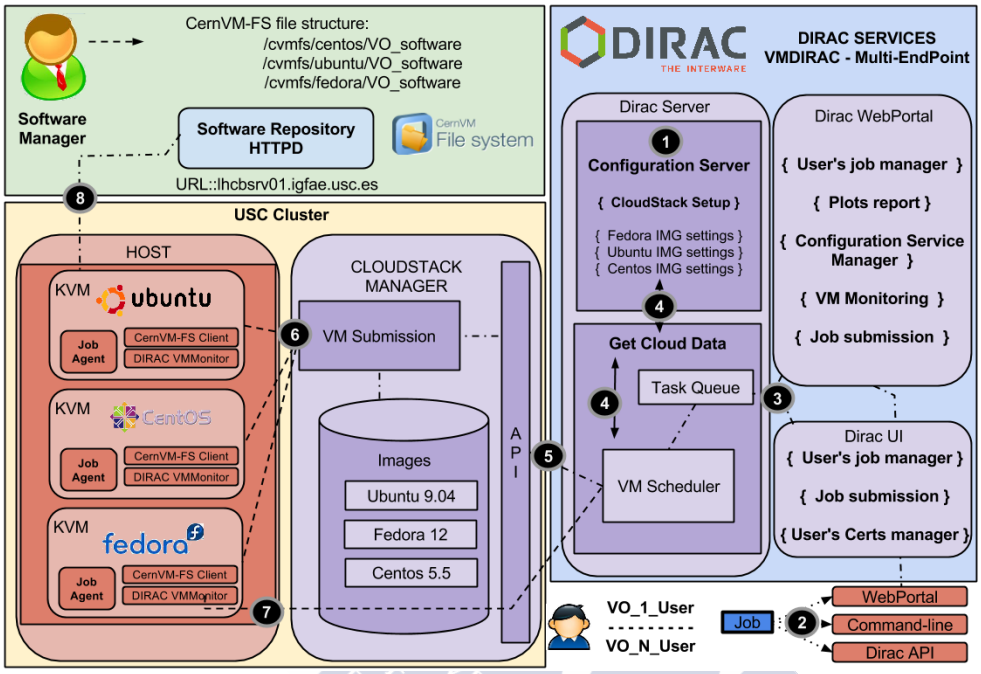


Figure 5.4: Multi-EndPoint prototype: USC CloudStack infrastructure Multi-VO and Multi-Platform tests.

The tests consisted in the submission of the simulations described in Section 3.2, in order to test the Multi-VO architecture with different instances, which were launched by one CloudStack 2.2.3 manager. The number of jobs submitted were around 500 of short jobs, as shown in Figure 5.5, and 50 long jobs, as shown in Figure 5.6.

Figure 5.5 shows 104 executed jobs with errors as "identified as stalled job", which means there is no contact with the job agent for a long time, 57 errors with "Application Finished With Errors", where the simulations had some failure in the process of execution, and 3 errors with no space in disk. The trace of "stalled jobs" failures showed they were due to the saturation of the hosts machines. On the other hand, 443 short jobs finished successfully of all the executed jobs.

Figure 5.7 shows the low efficiency of short jobs. This problem was solved for long jobs with the modification of the MaxVMs parameter, described in section 4.3.3, which is the maximum number of VMs allowed to run simultaneously. Figure 5.6 shows that the long jobs were executed successfully.

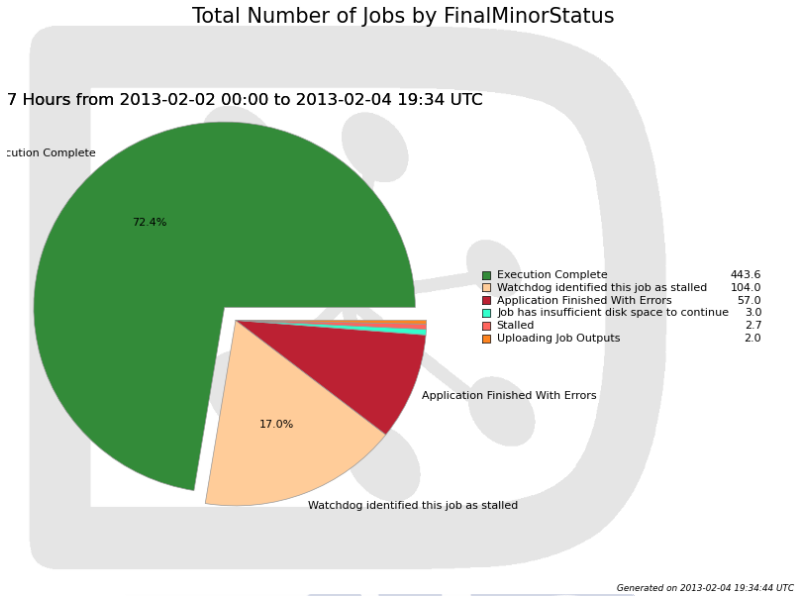


Figure 5.5: Executed short jobs by final minor status.

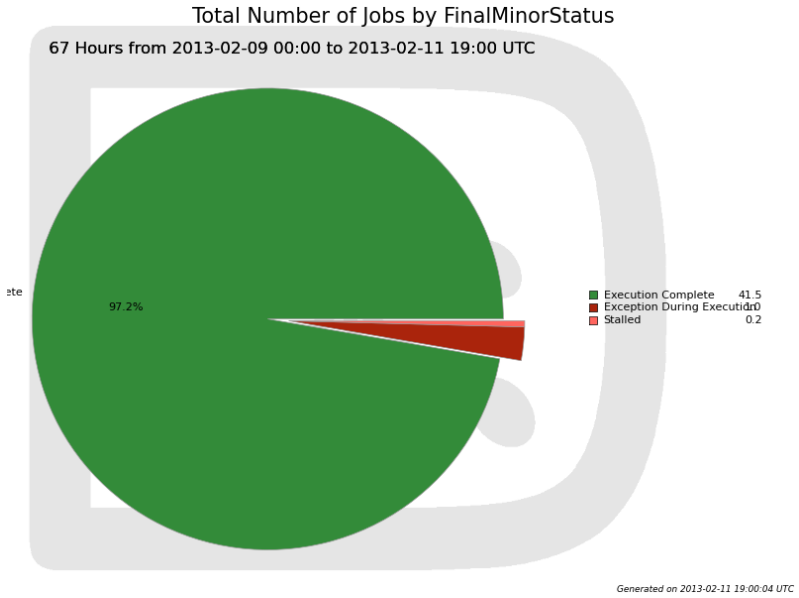


Figure 5.6: Executed long jobs by final minor status.

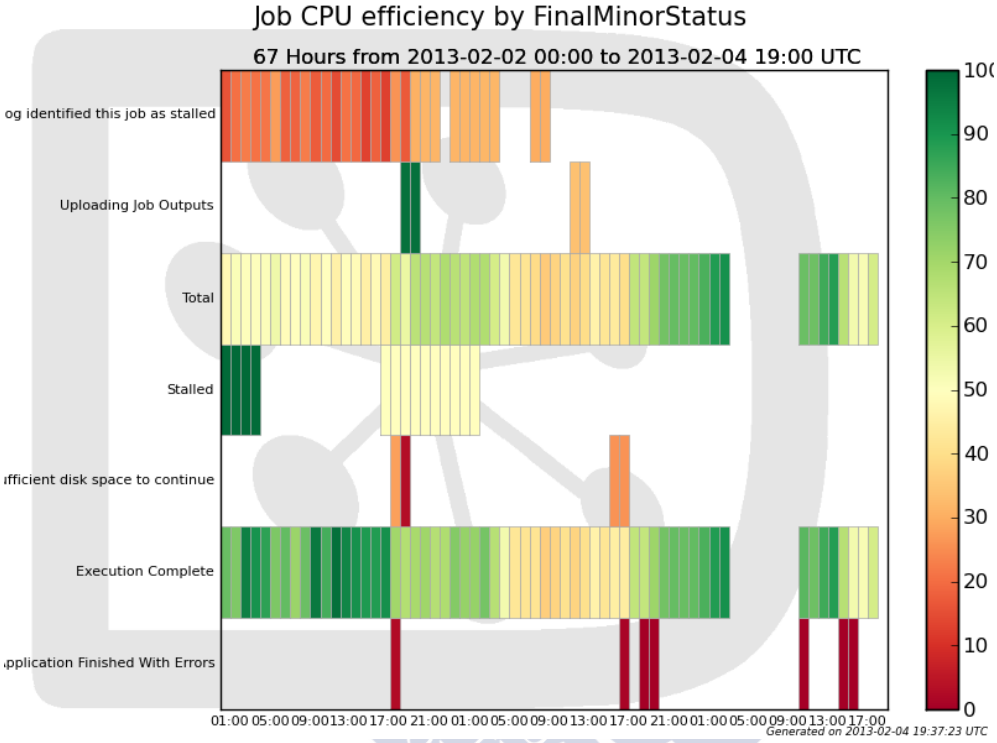


Figure 5.7: Executed short jobs CPU efficiency.

Figure 5.8 shows the running VM-monitoring with the VMs submitted to Cloudstack for long simulations. The top figure shows the average load of the simulations in the VMs at the left-side, and the simultaneously running VMs at the right-side. In the bottom figure at the right-side is shown the executed VMs separated by Platform, and shows a maximun of 8 Centos-VMs, 4 Fedora-VMs and 3 Ubuntu-VMs submitted at the same time.

In the next subsection the Multi-SITE test with PIC and USC is described, together with a couple of Cloud drivers for Cloudstack and OpenNebula managers.

5.2.2 Results of Multi-SITE for CloudStack and OpenNebula

This subsection includes the test results for Multi-SITES with the CloudStack and OCCI / OpenNebula, in order to evaluate the Multi-EndPoint design. The CloudStack testbed in

Figure 5.8: Running VMs monitoring.

this first integration was the virtualized CloudStack Network of the USC LHCb TIER-2. The OCCI/OpenNebula testbed of the DIRAC integration with OCCI/OpenNebula has been tested, using an LHCb application workflow. The main steps of the testing jobs are:

1. Environment login configuration **(LbLogin)**
2. Environment Gaudi and Gauss configuration **(SetupProject)**
3. Monte Carlo event generator for LHCb **(Gauss event generator application based on the Gaudi framework)**

The test job is therefore a scientific application with high CPU consumption and low I/O. The test infrastructure is the PIC Cloud testbed infrastructure, which is using OpenNebula Cloud manager and is deployed in 16 physical nodes. In this test, one node is used as OpenNebula Cloud manager server, implementing the OCCI server. The other 15 nodes are Cloud nodes with KVM hypervisor. Each physical node has 2 quad-core processors IntelXeon X5355 @ 2.66GHz each and total of 16GB memory.

A DIRAC service including the VMDIRAC extension integrating the OCCI/OpenNebula has been deployed. The virtual bootstrap image used for this tests is a cernvm-batch-node 2.4.0 [20]. OpenNebula instantiates VMs of many different sizes. For this test, small instances

with 1 core and 2 GB of memory have been chosen. This is considered a worst possible case scenario aimed to define a baseline of performances for comparison purposes.

This test is composed of 500 jobs of 100 events submitted to the DIRAC server. The DIRAC server is configured to run a maximum of 80 VMs on the OpenNebula Cloud infrastructure. Following the scheduling schema described in section 4.3.3, every 60 seconds, the VM Scheduler agent will check the TaskQueue, and if a matching job is found, it will submit a new VM. Thus, having all the workload on the TaskQueue at the beginning of the test, the ramping-up of 80 VMs and continuous execution of the DIRAC Jobs can be tested. When the 80 VMs have been started, a total of 10 Cloud nodes will be used, each running 8 VMs. One of the goals of the test is to obtain an overhead analysis of the OCCI/OpenNebula submission system and the HEPIX contextualization, and also, an overhead of the DIRAC machinery. To analyze the KVM hypervisor overhead, the metric of event/core/hour and the running job timestamps (walltime) were recorded and compared with the same metrics on jobs running in a non-virtualized environment. These are 50 jobs of 100 events directly submitted to CREAM CE [35], and running in a single Worker Node with exactly the same hardware that the Cloud nodes in the OpenNebula setup and with a standard WLCG production configuration (Scientific Linux 5 OS with a CVMFS client). In both tests, all the jobs were completed successfully without resubmission, and all the VMs were running without errors in the OpenNebula infrastructure. It is worth to mention that preliminary tests were done until a production ready state was reached. Some technical issues were faced. Additional configuration and tuning were done in different parts of the Cloud, which initially were affecting performances and degrading services causing errors, mainly regarding some optional packages of OpenNebula installation: the improved xmlparser and the nokogiri for html parsing. Also, other technical issues like the Sunstone SQLite problems when used with NFS, and the reliability of a shared or dedicated node for the Sunstone server, and the creation and deletion of the DIRAC Virtual Machine Instance.

The results of this test are shown in Figure 5.9. The ramping-up shows the OCCI/OpenNebula submission and HEPIX contextualization overhead within green-red lines. On the same ramp-up within red-black lines one can see the DIRAC running VM overhead with the real jobs timestamps. On the plateau, the VM submission line (green) and the VM running (red) are the same and equal to 80. At this point, all the VMs are submitted and running. Moreover, there is a DIRAC running VM overhead shown within green-black lines. The tail overhead in this test is about 15 minutes for each VM and can be configured on the DIRAC Configuration

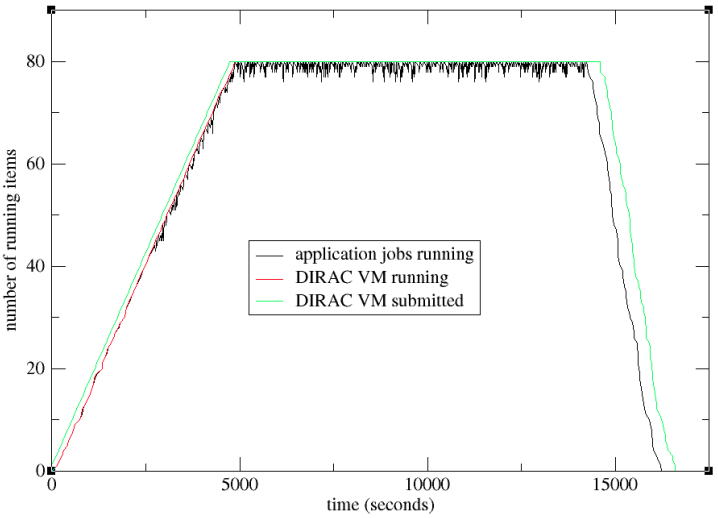


Figure 5.9: OCCI / OpenNebula Integration with DIRAC Test Results.

Server. Regarding the second metric of the test, aimed to obtain the KVM hypervisor overhead, the result in the Cloud testbed was 23 event/core/hour, while in the real Worker Node was 24.49 event/core/hour, which is about 6% of overhead in this worst scenario possible, with single core VMs.

The Multi-SITES test setup is a Monte Carlo LHCb simulation of proton-proton collision. 1000 jobs are submitted to DIRAC to be run on the Cloud, and these jobs will be submitted to CESGA, PIC and USC. Each job is using the CernVM-FS distributed software repository [30] of LHCb to run a simple workflow composed of LbLogin, SetupProject and Gauss for the creation of 100 simulated events. The used images are in the *OpenNebula end-points (CESGA and PIC)* the cernvm-batch-node 2.5 as golden image and a context image. CernVM images are including the CernVM-FS client. The same golden and context image are upload at CESGA and PIC, with the corresponding dynamic contextualization to each end-point indicated on the creation of the VMs. This contextual parameters are taken from the Configuration Server to provide the values of the VM network configuration and the CernVM-FS HTTP proxy to be used on the VM CernVM-FS client. In the case of the USC, the image is the CentOS 5.5 with a prepared DIRAC and a CernVM-FS client.

Figure 5.10: EndPoint prototype with Multi-SITE Cloud monitoring.

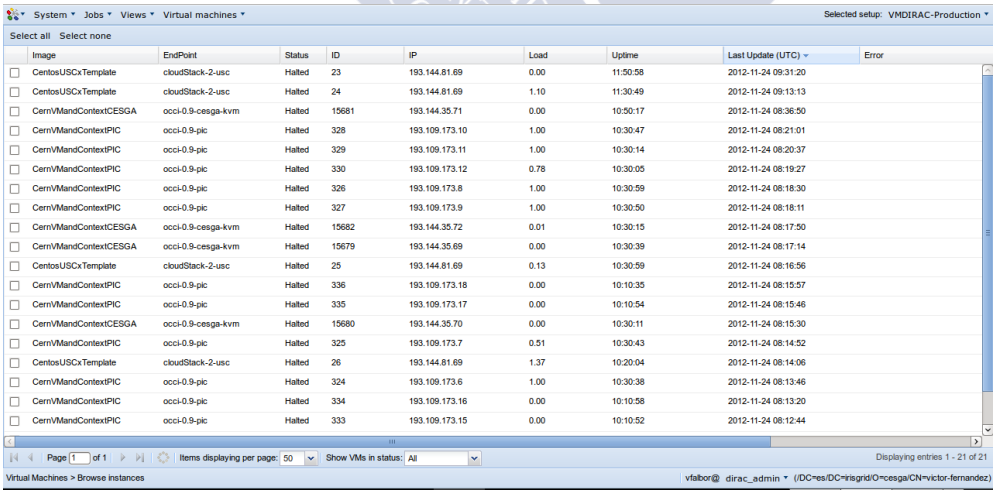


Figure 5.11: EndPoint prototype with Multi-SITE instances status monitoring.

The maximum number of VMs to be running simultaneously a this test are 21. It is just corresponding with the aggregation of the maximum number of VMs slices in the end-points (PIC 13 VMs, CESGA 4VMs and USC 4VMs). Every VM is an instance of 1 core and 2GB of memory. The DIRAC VM scheduler has been prepared to create a VM in each of the EndPoints, in case of jobs pending to be matched, in a loop of 1 min. Figure 5.10 shows the prototype test results. The plot shows the running VM per end-point. There is a fast ramping-up corresponding to the scheduler policy of the test, up to the maximum number of VMs per each end-point. The plateau is maintained for more than 10 hours of VM running, matching jobs about 25' running, but this is DIRAC specific. By the end of the test, the USC network connectivity was flapping. For this reason two of the USC jobs were unable to send heart-beat to the VM Manager, DIRAC was resubmitting these jobs and this is the reason of the USC queue of running VMs by the end of the test. Figure 5.11 shows in the Web Portal, the status of the instances during the execution of the test.

5.3 Results and Evaluation of Federated Hybrid Clouds Prototype (2013)

As a final result of the previous prototypes, a new architecture for Federated and Hybrid Clouds (Rafhyc) were created, and this section shows the results of a prototype based on this architecture. The test use case was a Monte Carlo LHCb simulation of proton-proton collisions in order to test this infrastructure with VMDIRAC architecture. 2000 jobs were submitted to DIRAC to be run on the federated Cloud. Each job is using the CernVM-FS distributed software repository of the LHCb [30] to run a basic workflow, which contains a sequence of environment initialization, software stack setup and the creation of 50 simulated collisions. OpenNebula end-point at Occi-PIC uses the cernvm-batch-node 2.6 as golden image and a context image which is generic for any OpenNebula IaaS. OpenStack end-point at Nova-IN2P3 [82] is also using a cernvm-batch-node 2.6 image, it includes the public key of the VMDIRAC server to allow a ssh contextualization of the VMs.

CernVM images are including the CernVM-FS client. In Occi-PIC and Nova-IN2P3 VMs are dynamically contextualized using the parameters of the Configuration Server. In the case of the CloudStack-USC, the image is the CentOS 5.5 with a preinstalled DIRAC, a CernVM-FS client and all the necessary context within the image.

VM Horizontal Auto-Scaling Setup

Horizontal auto-scaling is the property to create or halt VMs, depending on the workload. For this purpose, VMDIRAC has VM scheduling and stoppage policies associated to each end-point. The VM scheduling policy of the test is elastic and aims a fast ramping-up corresponding with the submission of one VM to every end-point in 1 minute intervals. It is also targeting a non exhaustive use of the available VMs slots. For this purpose the Running Pod configuration section has the CPUPerInstance parameter, which defines the minimal overall CPU of the DIRAC jobs waiting in the task queued to submit a new VM. The parameter is used for the tuning of the VM delivery elasticity. Therefore, a CPUPerInstance can be set to a longer time to use the available resources in a more efficient manner, saving creation overheads, and to a shorter time to setup an exhaustive use of the available resources aiming to finish the production in a shorter total wall time, but with higher resource costs for the additional overhead. In regular basis, CPUPerInstance reference values, from shorter to longer values, could be defined to:

1. Zero to submit a new VM with no minimal CPU in the jobs of the tasks queue
2. A longer value could be the average required CPU of the jobs as a compromise solution between VM efficiency and total wall time
3. A very large value to maximize the efficiency in terms of VM creation overhead, for the cases where the production total wall time it is not a constrain

The testing setup has a compromise CPUPerInstance value, to balance the VM efficiency and the total wall-time, it is set equal to the required CPU time for the job workload of 50 simulated collisions.

The VM stoppage policy of the test is elastic, which stops the VM if there were no more jobs running in the last VM halting margin time. The test defines 5 minutes of VM halting margin time.

The testing setup has 250 VMs as the maximum number of VMs to be running simultaneously on the IaaS aggregation: 10 VMs for Cloudstack-USC, 90 VMs for Occi-PIC and 150 VMs for Nova-IN2P3. The VMs use 1 vcpu and 2 GB of memory. The hypervisors are KVM, and they are hosted as follows:

1. Occi-PIC hosts are BL460c Blade with 8 cores Intel Xeon L5420, 2.50 GHz and a total of 16 GB of RAM

2. Nova-IN2P3 hosts are DELL Poweredge C6100 systems with 24 cores Intel Xeon X5675 3.07 GHz and a total of 96 GB of RAM

3. Cloudstack-USC hosts are AMD 6400MT with 16 cores AMD Opteron 6128 Magny-Cours 2.0GHz and a total of 16 GB of RAM

VM Horizontal Auto-Scaling Results

Figure 5.12 is a plot tool from the VMDIRAC as part of the VM Interface of the Rafhyc prototype. It shows the running VMs during the test. Figure 5.13 shows the corresponding breakdown for the different end-points.

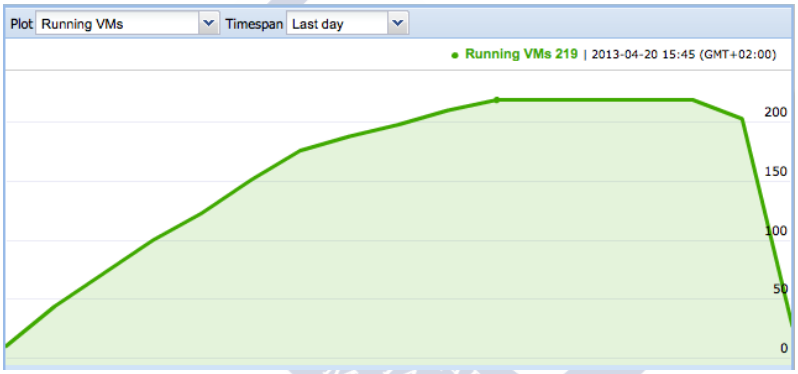


Figure 5.12: Running VMs in the 2k Jobs Test.

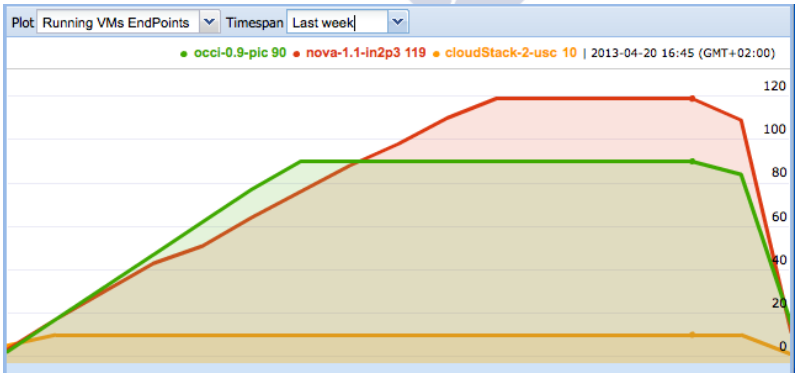


Figure 5.13: Running VMs by End-point in the 2k Jobs Test.

Figure 5.12 shows a total of 219 VMs on running status, at the beginning of the plateau (notice the little point on the green line corresponding to that time). In the same way, Figure 5.13 shows the information for the last part of the plain. One can see that Cloudstack-USC and PIC have reached the maximum number of VMs by end-point on the setup, 10 VMs and 90 VMs respectively. In the case of IN2P3, the maximum number of VMs has not been reached. Therefore, the scheduling policy of the setup is acting as expected, avoiding the use of all the VM slots. However, with a large enough workload in the DIRAC queue or smaller CPUPerInstance parameter, all the VMs would be submitted. This scheduling scheme, allows the tuning of each Running Pod for the available aggregated resources and the desired behaviour.

VM Performance Analysis

Further details of the virtualization overhead analysis (with the three components: VM creation, VM running and VM halt margin) in the context of separated single end-point submission to Amazon, CloudStack and OpenNebula can be found in previous works [57] [54] [100] and also some technical report about submitting multiple OpenNebula end-points in [99]. This thesis includes a VM performance analysis focused on some VM metrics:

1. VM Errors within Job Running Time
2. VM Creation and Context Errors
3. VM Creation and Context Time: The necessary time gap to boot and contextualize a VM up to reach the Running status
4. VM Runnable Uptime/Jobs: The VM Runnable time corresponds to the Running status, except the halt margin, which is the necessary time without running jobs on the VM to launch the VM halting (in charge of the VM Monitor Agent within the VM, and that in our test is set to 5 minutes). Number of Jobs corresponds to the total Jobs executed on a particular VM. Therefore, VM Runnable Uptime/Jobs is a metric of the efficiency of the VMs.

In the presented test there are no VM Errors within Job Running Time, all the jobs have been completed without errors and without any resubmission. For the metric VM Creation and Context Errors, there are no errors in Occi-PIC and Cloudstack-USC IaaS, but for the case of Nova-IN2P3 a total of 125 VMs were submitted, of which 6 VMs reached the Stalled status without having Running status. On the forensic analysis of these errors. Two different causes

were found. First, there was a single VM which took more than 30 minutes to be booted with an available network interface. 30 minutes is a configurable limit for the VM Manager to put a VM in Stalled status if no heart beat is received on that time. Second, on the scale-up test, 5 VMs where failing due to a bug found on the synchronization between the sftp copy of the scripts and necessary files, and the ssh run of the contextualization scripts. The bug has been patched.

The rest of this section, shows a Multiple Analysis of Variance, for the set of VMs which have run without errors, the 219 aggregated VMs, that are the statistic occurrences. The statistical design is a MANOVA 1-way, the dependent variables are the VM Runnable Uptime/Jobs and the VM Creation and Context Time, the independent variable (1-way) is the end-point with 3 categories: Cloudstack-USC, Nova-IN2P3 and Occi-PIC. Additionally, the submission time is considered for a histogram sequence. The submission time is a relevant concept on the Rafhyc architecture, because the optimization heuristic is considering different submission time windows with heterogeneous and dynamic performances.

That survey of a separate analysis of variance is shown on Table 5.1. The null hypothesis (H0) is VM Runnable Uptime/Jobs and VM Creation and Context Time, do not discriminate end-points. Both dependent variables have statistic results with $P < 0.01$ and $F > 1$, so they are significant and the H0 is not verified. Particularly, the separate variable analysis shows that the VM Runnable Uptime / Jobs is much more significant than VM Creation and Context Time to discriminate end-points.

Table 5.1: Separate Analysis of Variance Survey (2k Jobs Test)

	VM Runnable Uptime/Jobs	VM Creation and Context Time
Snedecor F	180.4290	7.8694
Probability P	0.0000	0.0005

Table 5.2 is a MANOVA survey, to consider the possible correlations between the two dependent variables. A multiple analysis of variance offers results of all variables instead of the individual results of the previous Table 5.1. The Canonical Vectors (CV) are projections of the original data in a lower dimensional space which allows a better discrimination between categories. CV 1 corresponds to VM Runnable Uptime / Jobs and CV 2 to VM Creation and Context Time. Two statistics are shown for each CV, the Wilk's Lambda and the Pillai's Trace, both are significant in terms of probability P and Snedecor F, in the same way as explained

above in the separate analysis. Wilk's Lambda statistic is close to 1 in the CV 2. At the same time, Pillai's Trace of CV 2 is close to 0, therefore, the conclusion is that VM Creation and Context Time are not correlated with the end-points in the overall analysis.

Table 5.2: MANOVA Survey (2k Jobs Test)

Exact Significances	Wilk's Lambda	Snedecor F	Probability P
CV 1	0.3049	58.1115	0.0000
CV 2	0.9335	7.6819	0.0006
Exact Significances	**Pillai's Trace**	**Snedecor F**	**Probability P**
CV 1	0.7397	42.2642	0.0000
CV 2	0.0664	7.6819	0.0006

As a summary of the separate and the overall analysis of variance, the two dependent variables, VM Runnable Uptime/Jobs and the VM Creation and Context Time, are significant to discriminate end-points, and in the case of VM Creation and Context Time such discrimination is indirect on the VM Runnable Uptime/Jobs.

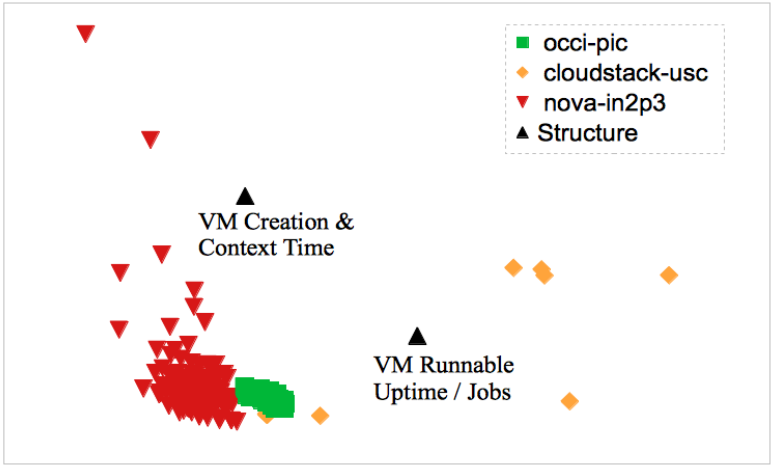


Figure 5.14: Biplot (2k Jobs Test): Correlations between Endpoints and Performance Metrics.

Further on the correlation analysis, we show a biplot in Figure 5.14. The biplot is the shadow in 2-dimensions of the normalized data, which maximizes variation. The measure-

ments are shown as one unit of each axis, to give the idea of observations plotted correlations, which correspond to the distances between the points of the plot.

Figure 5.14 plots the occurrences of the end-points together with the Structure of the two dependent variables. The first analysis of the biplot concerns the end-points discriminations. The three end-points are clearly categorized. VM submissions to Occi-PIC have a single category of occurrences. The occurrences of Cloudstack-USC are shown in a category on similar area than Occi-PIC, which are corresponding with the first VM submissions to Cloudstack-USC, and a group of few separated categories. In the case of Nova-IN2P3 a clear category is in a similar area than Occi-PIC and few occurrences are spread without a clear categorization. The previous MANOVA correlations are also shown in Figure 5.14, thus, VM Runnable Uptime Jobs is correlated with the main categories of the three end-points. On the other hand, VM Creation and Context Time could be correlated with some of the Nova-IN2P3 occurrences.

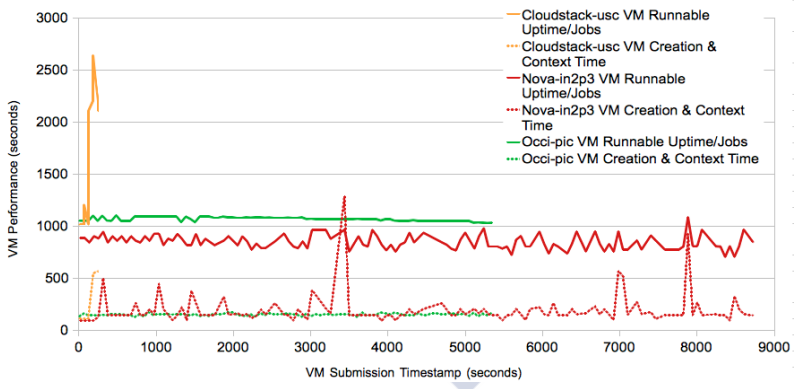


Figure 5.15: Histogram (2k Jobs Test): VM Metrics by End-point.

Figure 5.15 shows a histogram to evaluate how the variable correlations are changing along time. For this purpose, the performance metrics are plotted for each VM on the submission timestamp. The submission time is a parameter considered on the heuristic proposed on the Rafhyc approach, to manage the dynamic performance responses of the federated hybrid Cloud models. However, Figure 5.15 does not show large changes on performances within the end-points on temporal pattern. Anyway, this analysis is valid for short periods of time like this test, which is in magnitude of few hours. Previously to the analysis of the histogram, notice that in the time slice of the higher plateau of Figure 5.15, there was a total of 219 VMs.

In this moment, the overall load of the end-points, including VM for other purposes, was the following:

1. Cloudstack-USC: 56% of memory and 28% of CPU of the total available resources
2. Nova-IN2P3: 100% of memory and 53% of CPU of the total available resources . The hypervisor memory policy was tested, without a strict limit on the VM memory reservation on the physical memory of the hypervisor host.
3. Occi-PIC: 75% of memory and CPU of the total available resources

In this sense, the test shows reasonable scaling features. The dispersion on the Nova-IN2P3 performances would be related with the mentioned hypervisor memory policy in this end-point, which is being currently analysed by the SITE administrators and should be tuned. In these conditions, Figure 5.15 shows a clear categorization with the VM performances for the Cloudstack-USC end-point, which has a lower performance (larger values of the two metrics) for the second part of the submitted VMs. Moreover, there is a smooth decreasing trend on the Occi-PIC VM Runnable Up-time / Jobs and a larger dispersion on Nova-IN2P3 VM Runnable Uptime/Jobs. The response on VM Creation and Context Time of Nova-IN2P3 and Occi-PIC is as expected because the trends have no slope, which means that this end-points scales best.

An additional comment about the Nova-IN2P3 VM Creation and Context Time is the dispersion of responses. This is related to the time to start the VM network interface which is much higher than the VM booting time. Probably, this is related to the method used for the IP assignment, while at the other end-points the IP is assigned automatically. And this setup could also be adopted in Nova.

5.4 Results and Evaluation of Big Data Prototype (2014-2015)

This subsection describes the test results for BigDataDIRAC prototype, where the objectives were the proof-of-concept of the new DIRAC extension for Big Data software [52].

An architectural view of the implemented solution to integrate Big Data resources with DIRAC is shown in Figure 5.16. At the top level one can identify five main components: the user interface represented by a human figure, the DIRAC central servers running at the University of Santiago de Compostela (USC), with the DIRAC Interware logo, and three

institutions with four Hadoop clusters. BIGDATA.usc.es and BIGDATAHIVE.usc.es are the clusters at the University of Santiago de Compostela (USC) in Spain, BIGDATA.cesga.es is in the Galician Supercomputing Center (CESGA), also in Spain, and BIGDATA.ihep.cn is the cluster in the Institute of High Energy Physics (IHEP) in China. For better portability of the setup, all DIRAC components were created as virtual machines running in a CloudStack [32] cluster at USC. The Hadoop clusters were also created using cloud, using OpenStack [104] in the case of USC and IHEP, and OpenNebula [103] in CESGA. This last institution is also using an innovative system [95] for Hadoop Cluster creation, which allows the deploying of Hadoop clusters on demand [76] in seconds.

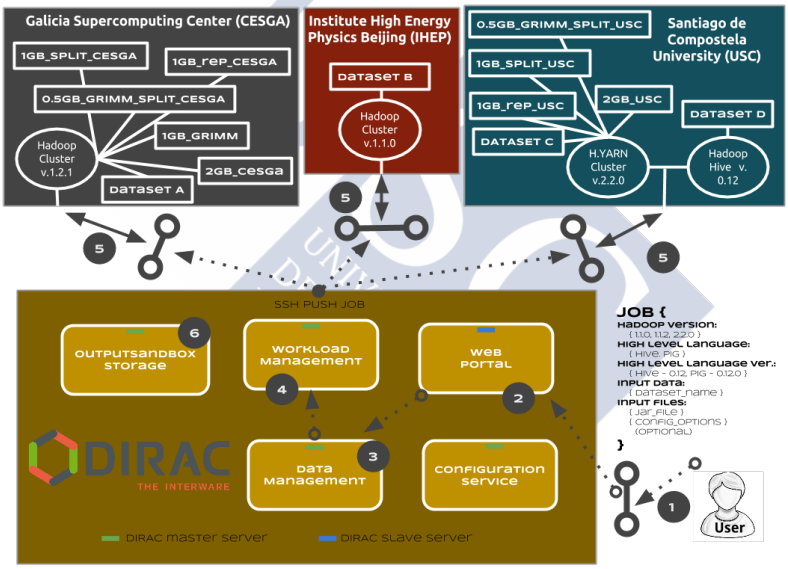


Figure 5.16: BigDataDIRAC Setup for testing the architecture.

Figure 5.16 shows the different steps a job goes through until it has finished. Initially, the user has to specify the basic settings for the job. In this specification the following parameters are required:

1. The Big Data software (only Hadoop is accepted at this moment, but new directors could be created for Spark [143], Twister [43], Phoenix++ [118], or other Big Data software) and version (the currently supported and tested versions for Hadoop are 1.1.0, 1.1.2 and 2.2.0, but other versions could work without problems).

2. The high level language used (if any) and its version (the 0.12 version of Hive have been tested in this testbed).

3. The dataset need to be added to the job. This datased should have been previously registered into the DIRAC file catalog by the DIRAC administrator (for production environments, a DIRAC agent could register automatically the new datasets associated to DIRAC.)

4. The input data associated to the job, such as, the jar file or Hadoop configuration files.

There are three different physical DIRAC machines at USC: the DIRAC main server (indicated with a small green rectangle in Figure 5.16), the DIRAC user interface (not shown in figure), and the Web Portal (blue rectangle), which allows the users to check the accounting and the job status. It also allows, if the user has the necessary privilege level, to manage the Big Data EndPoints connected with DIRAC.

Following with the steps of Figure 5.16, after submitting the job through the Web portal, the dataset is checked by the Data Management system, which ensures that the dataset is well formed before allowing a job to be created, with the pre-defined DIRAC form /vo/dataset_path. Furthermore it checks whether the dataset is registered in the File Catalog and, finally, it gets the SITE or SITES where the files or the replicas are stored. At this moment, the Workload Management system manages the job with the BigDataSchedulerAgent and BigDataMonitoringAgent. As it was described before, if the conditions are satisfied and the Director is selected, the job is submitted to the specific SITE in the step 5. In parallel, the Workload Management is constantly requesting the job status of the job. Once the job is finished, the BigDataMonitoring, in the case of normal job submission, or the InteractiveJobMonitoring, in the cases of Hive or Pig, gets the job output from HDFS and stores this output in the DIRAC OutputSandbox storage. During all this process, the user can monitoring the job status, and when the execution is finished, he can download the job output from the Web Portal.

Figure 5.16 also shows the datasets used in our experiments. The datasets 2GB_* and 1GB_split_* are randomly generated. The datasets 1GB_rep_* are replicas of the 1GB_split_* ones, in different location. The dataset 1GB_Grimm is a text file, containing books from the Grimm brothers, whereas the 0.5GB_Grimm_split_* datasets are splitted from the previous one. Furthermore, each of the datasets A, B and C contains 120 files of books in text format, of around 1.1 MB. Finally, the dataset D contains two tables, one with information of

patents (2,923,923 records) and a second one with the cites used by these patents (16,522,439 records).

Discussion of the proposed experimental setup

In order to evaluate the suitability of DIRAC integrated with Big Data software to perform Big Data analytic using Hadoop we have selected use cases representing typical small and medium-size jobs. These use cases use real data sets and a fairly common MapReduce operation, for which the setup described in the last section has been optimized. However, all the pieces involved are completely neutral and can be directly reused for other purposes with a different configuration. The strengths of the proposed setup are:

- Usage of a mature framework, DIRAC: the general functionality is well tested and there is no need to invest extra manpower on developing it. Many additional features are already built in like accounting or web portal that otherwise would take a long time to develop, test and setup. Once the use of Big Data resources is achieved, Grid, Cloud and local resources would seamlessly be available for a big community of users that could use different systems for software processing.

- Modularity: Different functional pieces are encapsulated and a plugin mechanism is used to ease their replacement when necessary. For instance, making use of a different Big Data provider just requires the definition of a new type of Big Data software with an associated module that knows how to handle the submission request to the new provider. Everything else remains the same.

- Efficient use of Resources: the proposed solution attempts to get access to computing resources only when there are waiting tasks matching the capabilities of the resource. Then, once the resource is accessed, it is used while there are pending tasks or the resource capabilities are not exhausted. This approach greatly reduces the overheads due to the resource reservation phase which is something intrinsic to any distributed computing system.

There are still some aspects that might need further development. For instance those related to the decision of queue limitation from the resource provider. In the current implementation, this decision is based on very few parameters: the cpu time requested by pending tasks, the total number of requested tasks, and still running, but the limits should be added by

the DIRAC admin in the CS by hand. Of course, if one can reach resources from different providers, a new algorithm based on availability, capabilities and cpu power, as well as the priorities of waiting tasks needs to be developed.

Use case tests and results

Description of real-world benchmarks and their results after the execution are included in this subsection. It is organized as follows: first, a proof-of-concept of BigDataDIRAC is deployed with common Big Data jobs executed in the different clusters. Second, it is presented a performance evaluation to test the BigDataDIRAC workload management [53], including three use cases corresponding to typical user job patterns.

Usecase: BigDataDIRAC proof-of-concept

This part covers the initial experience with the usage of small-size Hadoop and High level languages datasets from the four End-Points shown in the setup described in the beginning of the section 5.4.

In order to study the stability of the setup in different situations four campaigns were launched, in the first one, the typical Big Data use cases were launched, and around 300 jobs of different Hadoop versions and Hive were executed. This first use case, representing small-size Hadoop and Hive datasets, was based in splitted datasets of Grimm Brother's books, with around 1.1 MB in text format, for each SITE. In the case of Hive, 478 MB in two tables with patents and cites of this patents were used. Figure 5.17 shows the total number of jobs executed in the End-Points, in the case of Hadoop a word-count was launched and in the case of Hive a join query with both tables.

Figure 5.18 shows the total number of jobs by SITE, this plot shows that the SITE which has executed the higher number of jobs was BIGDATA.cesga.es with 54.4% of the jobs executed. Figure 5.19 shows the execution time of the jobs submitted and executed in the different EndPoints, this measure is obtained from the difference between the unix timestamp registered in DIRAC JobDB database measured when the user submitted the job and the time registered in JobDB when the job has finished. Some parameters of the BigDataMonitoring agent have to be tuned, such as the pooling time, in order to decrease the execution time registered in DIRAC, and also in the first half of the plot the jobs were executed with overloaded Hadoop clusters, and in the second half the tuning of the agents and the low load of the machines is showing a decreasing in the jobs execution time.

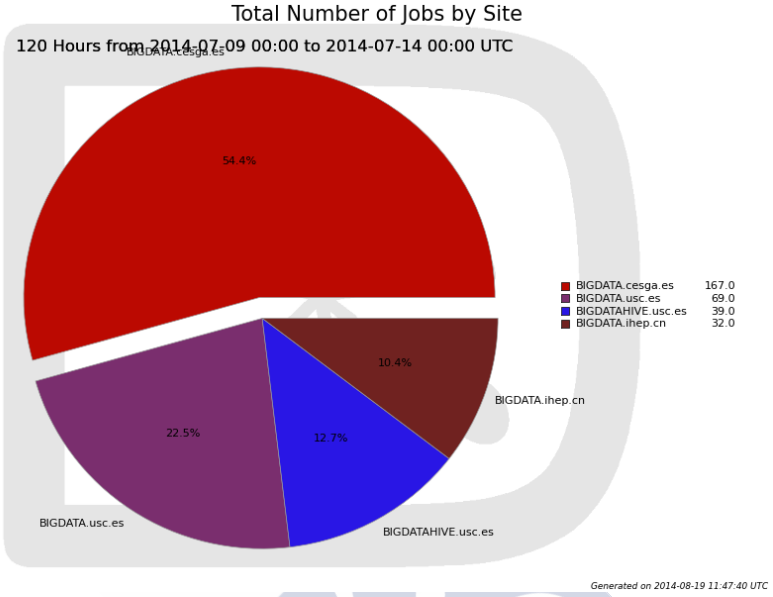


Figure 5.17: Total jobs of the first campaing by SITE.

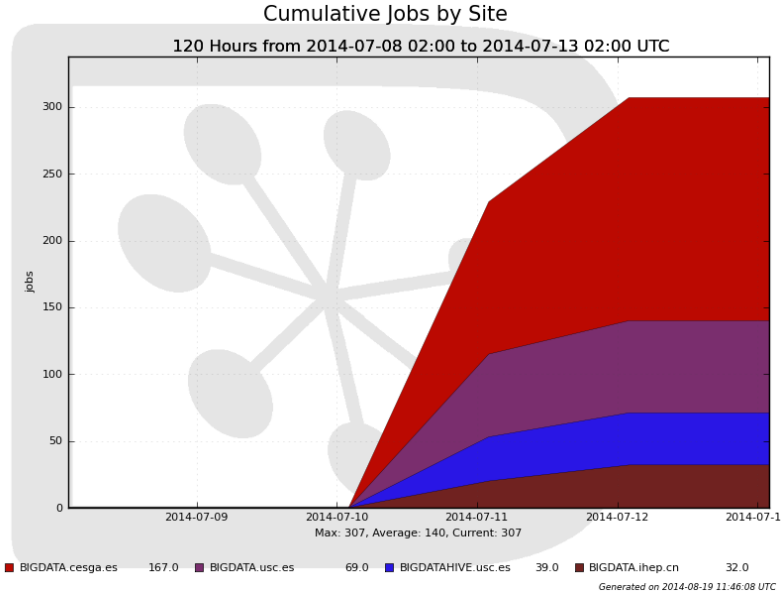


Figure 5.18: Cumulative jobs of the first campaing by EndPoint.

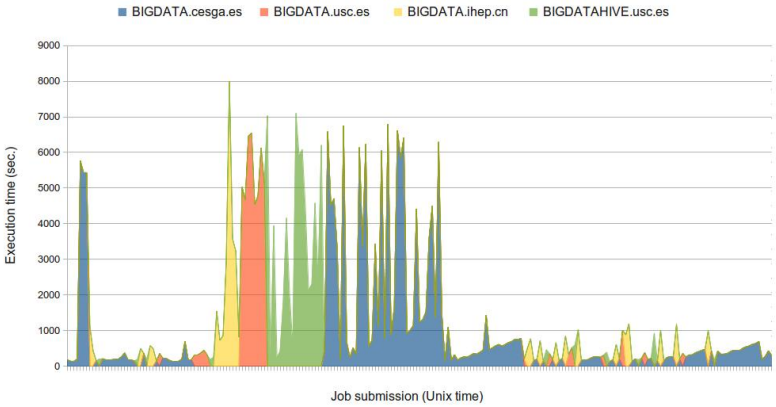


Figure 5.19: Total big data jobs grouped by EndPoint.

End-Point (SITE)	Wordcount (Sec.)	Task track.	Rep.	Blocks
BIGDATA.cesga.es	1123,55	5	3	118 (avg. block size 7424 B)
BIGDATA.usc.es	2314,09	1	1	118 (avg. block size 7424 B)
BIGDATA.ihep.cn	1404,88	1	1	118 (avg. block size 7424 B)

Table 5.3: Job average features of Wordcount MapReduce

Table 5.3 is showing the job average for the submitted jobs in this first campaign, also the number of data nodes and the HDFS block size of the HDFS datasets in the different EndPoints. Table 5.3 shows that in the case of BIGDATA.cesga.es makes sense a lower average wordcount, despite the fact that the block size is equal to the other two Hadoop clusters, because this cluster has a greater number of nodes than the others. And in comparison, the BIGDATA.usc.es and BIGDATA.ihep.cn, having the same number of data nodes and block size, the second offers a better performance. In the following tests, the execution of a higher number of jobs will be described in further detailed.

Table 5.3 is also showing the number of replicas, which define how many copies want to be stored of each block. Cesga is using three replicas that will allow different nodes to execute the same Map Reduce tasks-speculative execution-mitigating the performance degradation that could be experienced in a heterogeneous cluster with VMs running under hypervisors with different load conditions. In the case of others, there is only one dfs replica, which in

the case of several task trackers having just one replica eliminates the fault tolerance of the system, making it much less reliable, and with big size datasets could cause to consistently fail. The Map tasks were 121 and only one Reduce task for each EndPoint.

End-Point (SITE)	Join query (Sec.)	Task track.	Rep.	Blocks
BIGDATAHIVE.usc.es	2186,4	1	3	1 (avg. block size 78967873 B)

Table 5.4: Job average features of Hive job query

The features of USC Hive EndPoint (BIGDATAHIVE.usc.es) are showing in Table 5.4, that the average time was around 36 minutes for Hive Jobs. Three jobs were created with a total of four Map tasks and three Reduce tasks.

End-Point (SITE)	HDFS READ (bytes)	HDFS WRITTEN (bytes)
BIGDATA.cesga.es	890743	902218
BIGDATA.usc.es	892277	902218
BIGDATA.ihep.cn	889209	902218
BIGDATAHIVE.usc.es	269766417_Job_A	67882731_Job_A
	67883105_Job_B	736_Job_B
	1110_Job_C	440_Job_C
	337650632_Total	67883907_Total

Table 5.5: Read and write HDFS bytes in wordcount and Hive join query per job

As a result of this first campaign, at least 300 jobs were executed in three days continuously and 132.61 execution hours were consumed, since they were submitted until the output was uploaded to the DIRAC output sandbox. The wordcount jobs were launched to three different End-Points depending on the specific Input Dataset and they have read and written in total 231.94 MB and 234.89 MB respectively as is shown Table 5.5. Furthermore, Hive jobs have read a total of 10.06 GB of data and have written 2.02 GB in the HDFS. In conclusion, the DIRAC Big Data integration is prepared for more robust tests.

BigDataDIRAC workload management performance evaluation

The performance evaluation is testing the main hypothesis of Section 4.3.5: *federated BigDataDIRAC workload management is able to take advantage of aggregated computing power*

of distributed Big Data clusters by an efficient global load balancing. For this purpose, three usecases were tested to characterize typical user patterns: Big Data jobs with high IO throughput compared with CPU, jobs with balanced CPU and I/O consumption, and classical MapReduce jobs to filter lines including a searched substring, with higher CPU consumption compared to IO. Thus, a wide range of possible user jobs patterns were contemplated.

The three performance evaluation usecases are using a subset of the experimental setup described above, which is not including the IHEP Big Data cluster and USC Hive cluster, only the two others: USC and CESGA clusters. Each usecase tests the federated BigDataDIRAC results compared with single cluster equivalent results, to validate the main hypothesis.

Usecase: Randomwriter for high IO througput

A very extended performance test for Hadoop clusters is the *Terasort* [105]. This Java job is composed of a *randomwriter* run, followed by a *sort* of the produced outputs for the first run to complete a terabyte sorting benchmark. In our particular usecase, some modifications have been introduced. The benchmark was separated in two decoupled runs to keep record of the system performances for different use patterns. Therefore, the *randomwriter* usecase analyses the high IO throughput pattern and storing results in the DIRAC data catalog for future use of the produced files. The original *randomwriter* has also been modified to allow different file size productions, to run benchmarks in the gigabyte magnitude.

Two tests were run, both submitting a single *randomwriter* job per Big Data cluster. The first test produces a 2 GB random content file, splitting the file in two parts. The first part is created with a job running in BIGDATA.usc.es producing a 1 GB file, the second is another 1 GB file created in BIGDATA.cesga.es. Both jobs were submitted to DIRAC at the same time. Table 5.6 shows results for the split federated 2 GB production, including metrics of Hadoop jobs runtime (HRT), the corresponding DIRAC jobs response time (RT) and the makespan, which is the response time for both jobs submitted to DIRAC at the same time. Times are in seconds.

Big Data Cluster	File size	Hadoop runtime	Response time
BIGDATA.usc.es	1GB	2264	2424
BIGDATA.cesga.es	1GB	950	1029
Total Makespan	2GB		2424

Table 5.6: Federated BigDataDIRAC splited: Randomwrite 2GB

To compare this federated performance, another test was run for a complete 2 GB random write file in a single cluster, and repeated for USC and CESGA clusters to test different powers of the single clusters. This comparison test is shown in table 5.7, where the job response time corresponds to the 2 GB makespan of the federated case of Table 5.6.

Big Data Cluster	File size	Hadoop runtime	Response time
BIGDATA.usc.es	2GB	4364	4433
BIGDATA.cesga.es	2GB	1900	2052

Table 5.7: Single cluster: Randomwrite 2GB

As expected, *randomwrite* results obtained by federated BigDataDIRAC shows a performance constrained by the cluster with less power, in our case USC. With the tables results a general rule can be expressed as: given a *randomwrite* problem of size *S*, it can be splitted in *N* Hadoop clusters using BigDataDIRAC, obtaining a performance improvement of processing *lessPowerCluster(S/N)*. In this way, with an additional aggregated cluster of similar power than USC, the federated makespan would be about *4433/3*. So far, The hypothesis *federated BigDataDIRAC workload management is able to take advantage of aggregated computing power of distributed Big Data clusters by an efficient global load balancing* has been validated for this usecase, showing improvement by *lessPowerCluster(S/N)*. Actually, *S/N* performance is not exact, because there is some DIRAC overhead corresponding to the difference between Hadoop runtime and DIRAC response time. Analyzing the federated single jobs of 1 GB, it is shown overhead ratios of 7.1%(USC) and 8.3%(CESGA), while for 2 GB files production we have ratios of 1.6%(USC) and 8.0%(CESGA). Thus, generally, the DIRAC overhead is independent of the job size and is affected by the particular DIRAC and underlying Big Data resources conditions.

Usecase: Sort for balanced CPU and I/O consumptions

Sort usecase is using the previous DIRAC data catalog of the *randomwrite* to produce 1 GB sorted files. DIRAC catalog has two replicas for the unsorted files, one at CESGA and the other at USC. The test reads the input files, processes a MapReduce sort algorithm, and creates an output file. Total makespan included 10 repeated 1 GB sort jobs. Table 5.8 shows results for the replicated federated 1 GB sorting. In this case the federated strategy is to replicate the 1 GB dataset, instead of dataset spliting of the previous usecase. The 10 sort jobs are submitted to DIRAC at the same time, then each job would be run in the first free cluster found by the load balancing scheduling described in Section 4.3.5.

Big Data Cluster	File size	Hadoop runtime	Response time
BIGDATA.cesga.es	1GB	766	1104
BIGDATA.cesga.es	1GB	1520	1407
BIGDATA.usc.es	1GB	4344	10735
BIGDATA.cesga.es	1GB	1005	7424
BIGDATA.cesga.es	1GB	762	2540
BIGDATA.cesga.es	1GB	775	6733
BIGDATA.usc.es	1GB	4219	6038
BIGDATA.cesga.es	1GB	1459	3293
BIGDATA.cesga.es	1GB	1412	3984
BIGDATA.usc.es	1GB	2592	8424
Average		4696	1874
Total Makespan			10735

Table 5.8: Federated BigDataDIRAC replicated: Sort 1 GB

Table 5.9 shows the equivalent sort 1 GB test in a single cluster. For comparison reasons the most powerful cluster has been chosen.

Big Data Cluster	File size	Hadoop runtime	Response time
BIGDATA.cesga.es	1GB	756	808
BIGDATA.cesga.es	1GB	1428	1563
BIGDATA.cesga.es	1GB	747	14909
BIGDATA.cesga.es	1GB	1273	3710
BIGDATA.cesga.es	1GB	1191	5840
BIGDATA.cesga.es	1GB	1004	4430
BIGDATA.cesga.es	1GB	968	3002
BIGDATA.cesga.es	1GB	1425	2285
BIGDATA.cesga.es	1GB	766	13339
BIGDATA.cesga.es	1GB	1421	5152
Average		1097	5287
Total Makespan			14909

Table 5.9: Single Big Data cluster: Sort 1 GB

The average runtime for 1 GB sort jobs is 1,083s(CESGA) and 3,718s(USC). Sort power can be expressed as the inverse of the Hadoop runtime. Thus the theoretical power of the federated clusters would be the addition of the sort powers of CESGA and USC. Compared with the single CESGA power, the theoretical Hadoop improvement would be 29%, whereas

the actual improvement in the makespans was 39%. Therefore, the main hypothesis of the proposed approach is validated: *federated BigDataDIRAC workload management is able to take advantage of aggregated computing power of distributed Big Data clusters by an efficient global load balancing*, improving performance more than the simple theoretical power aggregation. This is due to the Hadoop semi-parallel features, which DIRAC can exploit when federating Big Data clusters: when increasing the federated clusters in *N* parallel workload is increasing greater than N.

Usecase: Grep for higher CPU than IO consumption

The *Grep* usecase is based in a 1 GB text, which is processed to obtain the lines which are including the search substring. Table 5.10 shows results for the split federated 1 GB Grep. In this case the federated strategy was to split the 1 GB dataset in two files of 0.5 GB one at CESGA and the other at USC. To process the grep, it is necessary to submit the job to both Big Data clusters, so two jobs (one in each cluster) are necessary to complete a single grep. The complete makespan is composed of 10 grep, corresponding to 20 DIRAC jobs, which were submitted at the same time and processed stochastically as described in Section 4.3.5. The results of a complete 1 GB grep are taken from sequential CESGA and USC job submissions, independently of when they were split processed. Table 5.10 summarizes the second DIRAC job of the pairs, including the runtime addition, the response time from the pair submission to the latest job end, and the total makespan from the larger DIRAC job response time. The concept was to test not only the replication processing strategy, as in the previous *sort* usecase, but also possible split processing strategies.

Table 5.11 shows the equivalent 1 GB Grep results for a single cluster submission in the most powerful cluster of CESGA.

Comparing the total makespan, the hypothesis *federated BigDataDIRAC workload management is able to take advantage of aggregated computing power of distributed Big Data clusters by an efficient global load balancing* has been validated with a performance improvement of 50%. However, the Hadoop runtime power per cluster, as explained in previous *sort* usecase, shows a theoretical power improvement of 84%, so in this case the theoretical power has not been improved in the total makespan. The reason of this is the stochastic job matching, which in the case of federated splitted pairs of jobs are not taking advantage of the fully parallel processing as a pair, obtaining lesser makespan improvement than might be theoretically expected.

Big Data Cluster	File size	Hadoop runtime	Response time
BIGDATA.cesga.es	0.5GB		
BIGDATA.usc.es	0.5GB	270	205
BIGDATA.cesga.es	0.5GB		
BIGDATA.usc.es	0.5GB	224	493
BIGDATA.cesga.es	0.5GB		
BIGDATA.usc.es	0.5GB	184	489
BIGDATA.cesga.es	0.5GB		
BIGDATA.usc.es	0.5GB	225	775
BIGDATA.cesga.es	0.5GB		
BIGDATA.usc.es	0.5GB	190	648
BIGDATA.cesga.es	0.5GB		
BIGDATA.usc.es	0.5GB	225	1034
BIGDATA.cesga.es	0.5GB		
BIGDATA.usc.es	0.5GB	262	1053
BIGDATA.cesga.es	0.5GB		
BIGDATA.usc.es	0.5GB	292	755
BIGDATA.cesga.es	0.5GB		
BIGDATA.usc.es	0.5GB	293	767
BIGDATA.cesga.es	0.5GB		
BIGDATA.usc.es	0.5GB	256	310
Average		242	652
Total Makespan			1104

Table 5.10: Federated BigDataDIRAC splitted: Grep 1 GB

Big Data Cluster	File size	Hadoop runtime	Response time
BIGDATA.cesga.es	1GB	225	276
BIGDATA.cesga.es	1GB	212	306
BIGDATA.cesga.es	1GB	222	1626
BIGDATA.cesga.es	1GB	208	1656
BIGDATA.cesga.es	1GB	143	606
BIGDATA.cesga.es	1GB	226	1056
BIGDATA.cesga.es	1GB	153	816
BIGDATA.cesga.es	1GB	225	788
BIGDATA.cesga.es	1GB	136	456
BIGDATA.cesga.es	1GB	153	1086
Average		190	759
Total Makespan			1656

Table 5.11: Single BigDataDIRAC splitted: Grep 1 GB

Chapter 6

Conclusions And Future Work

Motivated by the trend of Cloud, Big Data, and with the technological base of Grid computing, an integration was developed to allow scientific groups the use of different Cloud managers and Big Data software. New designs and components have been detailed, proving the DIRAC software extensibility, based in Open-source development standards, prototyping life cycle and re-engineering software. The coordination of different software appliances and middleware meet the nowadays beyond the frontline in the computer science state of the art, using for this purpose the DIRAC interware as the integration of different distributed middleware. The success of the tests shows different scenario of potential adoptions.

Virtual overhead metrics and simulations for different groups are also showing the viability of the Cloud technologies for scientific computing. The simulations were used to analyse statistically the performance over virtualization with different HEP software. One of the main challenges of this work is to know whether current Cloud technologies provide enough performance for specific machine configurations. In this sense, the Kernel-based Virtual Machine performance was analysed with the use of multivariate analysis of variance methods. Some performance problems were found in relation with hardware architectures, in a specific software called GAUSS of the LHCb experiment. The analysis of variance, including the MANOVA statistics, demonstrates the dependent variable walltime to have significant performance, for each software, of BM together with KVM, and there are significant differences depending on the length of the job. Furthermore, the amount of memory, when it is above 2 GB, does not seem to be an important factor in the performance of this job unlike the case of the number of cores, where it was observed an increase in performance as the number of

cores increases. The separated MANOVA of BM and KVM, presents similar results to the ones obtained in the previous case when MANOVA was applied to the six dependent variables, but the proximity of the centroids in the configurations of more core-memory shows more stability in Virtualized machines than in Baremetal.

As Cloud services gain maturity in HEP experiments, the correct configuration and setup of KVM hypervisor will be an important decision in order to reach an optimal performance in the specific HEP-Software.

Therefore, the work presented in this thesis proves that different groups may take advantage of the strong points of the Cloud technologies, in particular regarding the platform maintenance, the dynamic environment composition of the computing resources, and the hardware encapsulation on the computing nodes. A specific measurement of the major Cloud weak points was done: the performance overhead. We have shown that this is not an impediment for Cloud adoption in scientific computing. In this way, the design and the procedure of the software distribution has been detailed, and the successful coordination of different software appliances has been shown. The study case is based in previous tests and benchmarks and we have found an improvement in software distribution of CernVM-FS vs systems as NFS or HTTPd servers. This thesis shows the way to get CernVM-FS software repository access by normal users, with improvements in scalability, and avoiding problems as the high latency by using several level caching systems. The set-up of the infrastructure shows that Parrot can be an option in very simple cases, but when the complexity increases Parrot is unreliable. The described approaches make use of the flexibility provided by CernVM-FS, with some characteristics as the multi-platform client or the possibility of execution in Cloud environments.

Additional challenges of such adoptions, regarding the complexity and the heterogeneity of Cloud architectures were successfully faced in this work, with the use of the DIRAC interware, overlaying a wide range of the most used specific middleware in Grid and Cloud. First, CloudStack prototype was tested and implemented for the Formiga-Cloud project infrastructure. In the second prototype, tests were performed on the Multi-Site, Multi-VO and Multi-Platform concepts in different infrastructures. As a result, a new flexible architecture for constructing Federated Hybrid Cloud services was defined, Rafhyc is organized in Layers with a core Persistent Configuration and a set of components that can be adopted to fulfil the requirements of the particular classes of applications. The presented test results for the prototype implementation of a Rafhyc component subset demonstrates the capability of the architecture [51] to integrate aggregated resources for scientific computing and efficient im-

age management. The analysis of the variance of the test, including MANOVA statistics, demonstrates that dependent variables are significant in the end-point discrimination, with heterogeneous results depending on the different end-points conditions, which justifies the necessity of an heuristic approach to deal with such problem domains, such as the one given by Rafhyc. As federated Cloud services gain maturity, Rafhyc adopters may automatize the Federated Service Polling to provide the necessary information to implement the Computing Efficiency and Price Optimization and reach advanced levels of resilient resource management.

Finally, this thesis also shows how to aggregate Big Data resources in a federated context with DIRAC as the connector framework. A setup based on the DIRAC interware for distributing computing has been described and implemented, with new components to handle the use of Big Data resources running on Cloud clusters. With these components, Big Data resources can be integrated with other computing resources such as Grid or Cloud. The setup was implemented and tested using four Big Data clusters, with several benchmarks in a testbed that have federated three institutions in two countries, Santiago de Compostela University (Spain), Galicia Supercomputing Center (Spain) and the Institute of High Energy Physics (Beijing). A validation process with different phases was done, in order to verify the stability and the reliability of the solution. In a first phase three hundred jobs were executed in four clusters of the three institutions with small and medium-size datasets, using the equivalent in hours of five days running uninterruptedly. Almost 10.20 GB HDFS reading data and 2.33 GB HDFS writing data have been produced and successfully stored at HDFS and DIRAC OutputSandbox. In a second phase, three job patterns have been evaluated, the high IO, balanced CPU and IO, and high CPU were the benchmarks created for the performance evaluation. The benchmarks results have processed 46 GB of data with different medium-datasets of 0.5 GB, 1 GB and 2 GB. The successful execution of these tests have shown that with DIRAC and Big Data it is possible to take advantage of aggregated computing power, showing an efficient global load balancing scheduling.

This thesis shows that the integration of DIRAC and Big Data ecosystem in federated context is possible and it has been validated. The code used by DIRAC is open-source and is shared in public git repositories [39] [131] [15].

Future work in the roadmap is including the creation of a complete Cloud multi-scheduler, the integration with Hadoop tests with big-size datasets and real world datasets, the aggregation of Hadoop Streaming for running jobs written in different languages, like Python or Perl.

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Resumen

Cada día crece más la demanda de recursos de computación requeridos por los investigadores, capacidades de cálculo que coexisten con el creciente volumen de datos generados actualmente. Estos investigadores están a demandar un servicio de Computación de Altas Prestaciones (HPC) que permitan la ejecución de sus simulaciones a través de la computación distribuida para poder acceder a los máximos recursos posibles, facilitando este acceso de la forma más cómoda y segura para ellos. El término computación distribuida se utiliza para describir cualquier tipo de operación que implique un sistema distribuido, es decir, varios equipos que se comunican a través de una red. En este sentido, los sistemas distribuidos incluyen desde sistemas fuertemente conectados, como un supercomputador paralelo o un cluster, a débilmente acoplados, como en el caso de Grid Computing. El cluster implica un conjunto de nodos casi idénticos próximos entre sí, conectados a través de Gigabit o redes de fibra óptica. En el caso de la computación Grid puede ser visto como la federación de recursos de cómputo. Estos recursos generalmente están débilmente acoplados, son heterogéneos y geográficamente dispersos. El Grid Computing puede definirse según Ian Foster como "la capacidad de habilitar el uso compartido de recursos para resolución de problemas en organizaciones virtuales de carácter dinámico y multi-institucionales". Así, el Grid proporciona una infraestructura de computación distribuida que permite el uso de recursos de cómputo a múltiples organizaciones virtuales (VO), donde se entiende como VO al conjunto dinámico de personas o instituciones distribuidas físicamente. Este organización virtual suele estar sujeta a un conjunto de reglas y condiciones en la gestión de dichos recursos.

Dentro del trabajo que presenta esta tesis, se explora un nuevo enfoque para el diseño de sistemas distribuidos, centrándose en una mezcla de diferentes tecnologías Grid, junto con otras tecnologías novedosas en el campo de la computación distribuida como es el caso del Cloud o las herramientas para procesamiento en el área del Big Data. Todo ello se intentará

combinar con una solución escalable para aprovisionamiento del software específico de las organizaciones virtuales. Dentro del proceso de planificación y creación del nuevo enfoque, se describe también el conjunto de software intermedio encargado de conectar los recursos. Este software es necesario debido a que, por lo general, los recursos mencionados estarán físicamente distribuidos, y pertenecerán a diferentes organizaciones que podrían poseer posibles barreras políticas o administrativas, como es el caso de Universidades en donde las políticas de acceso a recursos internos pueden ser bastante restrictivas. Hoy en día, estas Universidades poseen redes conectadas con los centros de investigación con una velocidad y fiabilidad que posibilitan la ejecución de trabajos de cálculo científico de manera distribuida, y disponen tanto de recursos de cómputo existentes en aulas de informática utilizados para docencia, laboratorios, etc., como clusters de ordenadores pertenecientes a grupos de investigación, que en muchas ocasiones están infrautilizados en épocas de baja demanda. El uso de tecnologías distribuidas como el Grid o el Cloud, parte del trabajo de esta tesis, permitirá que estos recursos computacionales heterogéneos puedan ser reutilizados por los investigadores para realizar simulaciones, aportando una mayor cantidad de cómputo a la ya existente y deslocalizando los recursos entre distintos lugares alrededor del planeta. Para ello, se contará con soluciones software como es el caso de DIRAC, un software open-source creado como herramienta Grid dentro del proyecto LHCb, y que hoy en día es utilizado por un montón de organizaciones como es el caso de la Iniciativa Grid Europea (EGI) o el sistema de envío de trabajos al Grid en Francia (France-Grilles). DIRAC funciona como un sistema distribuido con agentes que permiten la utilización de distinto tipo de recursos de cómputo para la ejecución de tareas de cómputo de los investigadores. Otras herramientas que serán relevantes para la creación del nuevo sistema serán los gestores Cloud. Para la selección de los sistemas Cloud más adecuados fue necesario hacer un estudio que recogiese los más idóneos para la infraestructura que se quería crear, y algunos de los que se han elegido son bien conocidos como es el caso de CloudStack u OpenNebula. Y por último, en la integración de aplicaciones provenientes del campo del Big Data, se ha utilizado la herramienta de MapReduce Hadoop junto con otras del ecosistema como es el caso de Hive o Pig. Y en cuanto al tema de aprovisionamiento de software entre distintas plataformas, CernVM-FS constituirá la base de la solución de distribución de software presentada en esta tesis.

Por lo tanto, el objetivo de esta tesis podría sintetizarse como la creación de un sistema de envío de trabajos, que permita ser utilizado por varias organizaciones virtuales, basado en un software creado para funcionar en entornos Grid, pero que se adaptará para trabajar con

entornos basados en Cloud Computing y el Big Data. Y que contará con un repositorio de software centralizado para proporcionar el software necesario para la ejecución de aplicaciones. Puesto que el uso de recursos no dedicados es un requisito, el mecanismo propuesto deberá de realizar la asignación entre clusters no dedicados y dedicados y deberá de proporcionar diferentes plataformas dependiendo del grupo de usuarios que necesiten ejecutar trabajos. En otras palabras, el sistema deberá de permitir enviar trabajos de investigadores a Clouds federados y aulas de ordenadores de organismos, adaptándose a las políticas de acceso que hayan sido definidas. Deberá ser capaz de utilizar la plataforma más apropiada para su software, y adaptarse a los requisitos del proyecto Formiga-Cloud, que busca la creación de sistemas de envíos de trabajos en instituciones con recursos dedicados y no dedicados. Por lo tanto se pueden considerar como objetivos principales de esta tesis los siguientes:

- La integración de recursos dedicados y no dedicados, tales como salas de ordenadores en universidades y colegios. Esto permitiría aumentar la potencia de cálculo, especialmente por la noche y los fines de semana, cuando no hay actividad académica.

- La inclusión de una solución escalable para la distribución de software. Necesario para lograr una baja latencia y maximizar la eficiencia en el uso de los recursos.

- Una gestión sencilla de la infraestructura y de la herramienta que permita el envío de trabajos. Esta herramienta deberá de permitir el uso por varias comunidades de usuarios. Esto facilitará que los investigaciones se centren en su trabajo en lugar de preocuparse por las características específicas del hardware y software en las instalaciones donde se ejecutan los trabajos.

- Cada grupo de usuarios (Organización Virtual) tendrá una plataforma específica, software y prioridades de los usuarios para ejecutar su trabajo.

- La integración de software de Big Data como Hadoop bajo entornos Cloud para la ejecución de los trabajos con que permitan la ejecución de conjuntos de datos distribuidos.

- Posibilidad de incorporar indistintamente recursos Grid, Cloud o clusters Big Data cuando sea necesario y de proveedores externos.

- La incorporación de múltiples gestores de Cloud. El entorno Cloud con el que se inicia la creación del sistema estará basado en CloudStack, ya que se supone que es barato,

fácil de usar y administrar. Pero será necesario que la solución propuesta permita la integración del mayor número de gestores Cloud posibles.

Para la creación de todo el sistema, este trabajo comenzó con la obtención minuciosa de los requisitos necesarios, para pasar después a un proceso de integración básica. Posteriormente se estudió la optimización de un conjunto representativo de paquetes de software científico ejecutado en entornos Cloud. Para ello, fue necesario la realización de estudios estadísticos lo más próximos posible a los entornos en producción para poder determinar y crear las infraestructuras adaptadas evitando así la pérdida de rendimiento de los trabajos en la utilización de los recursos Cloud. Por último en el proceso de desarrollo se creó y probó el sistema de envío de trabajos que requerirá de grandes volúmenes de datos distribuidos, haciendo uso de software Big Data. Como resultado de todo este trabajo se diseñaron nuevos sistemas y componentes, detallados a lo largo de los capítulos de la tesis. El diseño y la versatilidad de las soluciones empleadas demuestran la capacidad de extensión de DIRAC, que está basado en normas de desarrollo Opensource. Estas extensiones creadas para DIRAC han sido desarrolladas haciendo uso de prototipos de ciclo de vida y técnicas de reingeniería de software.

Dentro del proceso de validación de los nuevos sistemas y componentes, se crearon métricas y simulaciones que han permitido demostrar la viabilidad de las tecnologías Cloud en la computación científica. Dichas simulaciones se utilizaron para analizar estadísticamente el rendimiento del software que se ejecutó en los entornos virtualizados y con respecto a entornos sin virtualizar. Así que uno de los principales desafíos de este trabajo fue conocer si el Cloud proporcionaba el suficiente rendimiento en configuraciones de máquinas virtuales típicas. Como resultado, a raíz del análisis estadístico se encontraron algunos problemas de rendimiento relacionados con arquitecturas hardware, en la utilización de un software específico denominado GAUSS utilizado en el experimento LHCb del Centro Europeo para la Investigación Nuclear o CERN. Y en cuanto al estudio estadístico, se demostró que el tiempo de CPU de las simulaciones dentro del Cloud, posee significancia pero esta es dependiente del software que se ha analizado. Se demostró que hay diferencias significativas en función de la duración del trabajo. Por otro lado la cantidad de memoria, cuando es superior a 2 GB, no parece ser un factor importante en el rendimiento de este tipo de trabajos a diferencia del número de nucleos de CPU que se empleen, donde sí se observó un aumento en el rendimiento proporcional al aumento de los mismos.

Por lo tanto, como parte de la conclusión de esta tesis se puede aseverar que así como los servicios en el Cloud adquieran madurez en la ejecución de simulaciones en el campo de la física de altas energías, la correcta configuración de hipervisores del tipo KVM será una decisión a tener en cuenta importante con el fin de alcanzar un rendimiento óptimo con este software específico de física de altas energías. Se ha demostrado que diferentes grupos de investigadores en el área de física de altas energías pueden aprovecharse de las oportunidades de las tecnologías Cloud, optimizando su rendimiento.

En cuanto a la parte de distribución de software de distintos grupos de usuarios con la herramienta CernVM-FS, a través de pruebas comparativas se ha demostrado que se mejora sustancialmente, con respecto a los otros sistemas analizados como es el caso de NFS o servidores HTTP que proveen del software a través de internet. Estos estudios comparativos permitieron el diseño y creación de un procedimiento de distribución de software detallado. En esta tesis se muestra la forma de obtener software accediendo a un repositorio de CernVM-FS, con mejoras en la escalabilidad y evitando problemas como la alta latencia utilizando varios sistemas de almacenamiento en caché a distintos niveles. Dentro del diseño del procedimiento para la distribución del software, se analizan los sistemas oportunistas como es el caso del software denominado Parrot, que interpreta el protocolo utilizado por CernVM-FS. Los resultados muestran que este software puede ser una opción en casos muy sencillos, pero cuando la complejidad aumenta es poco fiable.

Dentro del proceso de desarrollo prototipado, se realizaron y evaluaron varios prototipos, que van escalando en complejidad y se adaptan a los requerimientos propuestos inicialmente. En el primero de los prototipos que permitió una integración de DIRAC y CloudStack fue probado junto con la infraestructura del proyecto FormigaCloud. En el segundo, se realizaron pruebas sobre los conceptos multi-sitio, varias organizaciones virtuales y multiplataforma. Y como resultado, se definió una nueva arquitectura flexible para la construcción de servicios federados con cloud híbridos. Esta arquitectura se denominó Rafhyc y se organiza en capas. Los resultados de las pruebas realizadas sobre Rafhyc demuestra la capacidad de la arquitectura para integrar los recursos agregados para cálculo científico así como una gestión eficiente de imágenes. Y dado que los servicios de Cloud federados ganan madurez, adopciones como Rafhyc permitirán automatizar los servicios federados al proporcionar toda la información necesaria para aplicar eficientemente sistemas como la optimización de precios y obtener recursos a un nivel más avanzado.

Finalmente, en esta tesis también se ha diseñado, implementado y probado la integración de un software Big Data como es Hadoop con DIRAC en la agregación de recursos federados. Durante la creación de la arquitectura fueron creados nuevos componentes, que permiten que clusters Big Data puedan ser integrados de una manera sencilla con otro tipo de recursos como es el caso de los clusters Grid o Cloud. Para las pruebas de agregación se utilizaron cuatro conjuntos de datos distribuidos entre distintos centros de cálculo. Este sistema federado constaba de tres instituciones en dos países distintos, entre los que se encontraban la Universidade de Santiago de Compostela (España), el Centro de Supercomputación de Galicia (España) y el Instituto de Física de Altas Energías (Beijing). Para testear la agregación federada de recursos se diseñó un proceso de validación con diferentes fases, con el fin de verificar la estabilidad y la fiabilidad de la solución en la primera de ellas, en donde se ejecutaron trescientas simulaciones en cuatro clusters de Big Data de las tres instituciones mencionadas con volúmenes de datos de tamaño pequeño y medio, utilizando el equivalente en horas o cinco días sin interrupción, con casi 10,20 GB en lectura de datos en el sistema de archivos de Hadoop (HDFS) y 2,33 GB en escritura sobre HDFS. Una vez finalizada la primera base en donde se comprobó la estabilidad, en una segunda se evaluaron tres características, el alto IO, el balanceo de carga de CPU y IO, y por último el uso de CPU en condiciones de carga elevada. Los resultados, que procesaron del orden de 46 GB de datos con diferentes fuentes de entrada de tamaño medio 0,5 GB, 1 GB y 2 GB, demostraron que con DIRAC y Big Data es posible aprovechar la potencia de cálculo agregado, y que es eficiente en cuanto al balanceo de carga, demostrando así que la integración de un sistema Grid como DIRAC y el ecosistema de Big Data en un contexto federado es posible.

Como trabajo futuro en la hoja de ruta está incluir la creación de un multischeduler en Cloud, la integración con Hadoop con conjuntos de grandes volúmenes de datos de aplicaciones reales, y la inserción de Hadoop Streaming para ejecutar trabajos escritos en diferentes lenguajes de programación, como Python o Perl.

Bibliography

[1] About Globus Toolkit. http://www.globus.org/toolkit/about.html. Accessed at Apr. 2015.

[2] Amazon Elastic Compute Cloud. http://www.aws.amazon.com/ec2/. Accessed at Apr. 2015.

[3] Amazon Simple Storage Service. http://aws.amazon.com/s3. Accessed at Apr. 2015.

[4] Apache CloudStack. http://cloudstack.apache.org/. Accessed at Apr. 2015.

[5] Apache Hadoop. http://hadoop.apache.org. Accessed at Apr. 2015.

[6] Apache Hadoop HDFS Federation. https://hadoop.apache.org/docs/r2.6.0/hadoop-project-dist/hadoop-hdfs/Federation.html. Accessed at Apr. 2015.

[7] L. Apolinario, N. Armesto, and L. Cunqueiro. Background subtraction and jet quenching on jet reconstruction.

[8] L. Apolinario, N. Armesto, and L. Cunqueiro. An analysis of the influence of background subtraction and quenching on jet observables in heavy-ion collisions. 2012.

[9] M. Armbrust, A. Fox, R. Griffith, et al. A view of Cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[10] AWS Elastic MapReduce. `http://aws.amazon.com/elasticmapreduce/`. Accessed at Apr. 2015.

[11] M. Baker, R. Buyya, and D. Laforenza. Grids and Grid technologies for wide-area distributed computing. *Software: Practice and Experience*, 32:1437–1466, 2002.

[12] P. Barham, B. Dragovic, K. Fraser, et al. XEN and the art of virtualization. *SOSP Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, 2013.

[13] G. Bell and J. Grey. What's next in high performance computing? *Communications of the ACM*, 4555(2), 2002.

[14] V. Berzins. Software prototyping. In *Article Encyclopedia of Computer Science*, volume 4, pages 1636–1638, 2003.

[15] BigDataDIRAC, github repository. `https://github.com/vfalbor/BigDataDIRAC`. Accessed at Apr. 2015.

[16] J. Blomer, P. Buncic, et al. The CernVM file system. Technical report. Accessed at Apr. 2015.

[17] J. Blomer, P. Buncic, et al. Status and future perspectives of CernVM-fs. In *Congress of Computing in High Energy and Nuclear Physics*, 2010.

[18] J. Blomer, P. Buncic, and P. Fuhrmann. Towards Cloud computing for HEP using CernVM-fs. In *Gentner Day*, 2010.

[19] F. Brochu et al. Ganga: User-friendly Grid job submission and management tool for LHC and beyond. In *Journal of Physics: Conference Series*, volume 219. IOP Publishing, 2010.

[20] P. Buncic. CernVM - a virtual appliance for LHC applications. In *Proceedings of the XII. International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, volume 12. PoS Publishing, 2008.

[21] P. Buncic, C. Aguado Sánchez, J. Blomer, et al. CernVM: Minimal maintenance approach to the virtualization. In *Journal of Physics: Conference Series*, volume 331, page 052004. IOP Publishing, 2011.

[22] P. Buncic, C. Aguado Sánchez, J. Blomer, et al. A practical approach to virtualization in HEP. *Future generation computer systems*, pages 2190–5444, 2011.

[23] M. Cacciari, G. P. Salam, and G. Soyez. Fastjet user manual. Technical report, Cern, 2011.

[24] E. Capriolo, D. Wampler, and J. Rutherglen. *Programming Hive*. O'Reilly, 2012.

[25] A. Casajus and R. Graciani. LHCb DIRAC team. In *Journal of Physics: Conference Series*, volume 219, page 042033. IOP Publishing, 2010.

[26] A. Casajús and R. Graciani. DIRAC distributed secure framework. In *Journal of Physics: Conference Series*, volume 219, page 042033. IOP Publishing, 2010.

[27] A. Casajús, R. Graciani, S. Paterson, A. Tsaregorodtsev, et al. DIRAC pilot framework and the DIRAC workload management system. In *Journal of Physics: Conference Series*, volume 219, page 062049. IOP Publishing, 2010.

[28] C. Catlett and L. Smarr. Metacomputing. *Communication of the ACM*, 35(6):44–52, 1992.

[29] CernVM. `http://cernvm.cern.ch/portal/filesystem`. Accessed at Apr. 2015.

[30] CernVM-FS. `http://cernvm.cern.ch/portal/`. Accessed at Apr. 2015.

[31] A. Chierici and R. Veraldi. A quantitative comparison between XEN and KVM. In *Journal of Physics: Conference Series*, volume 219, page 042005. IOP Publishing, 2010.

[32] CloudStack. `http://www.cloudstack.com`. Accessed at Apr. 2015.

[33] CloudStack Dev. Doc. `https://cloudstack.apache.org/docs/en-US/Apache_CloudStack/4.1.1/html/Developers_Guide/`. Accessed at Apr. 2015.

[34] G. Collaboration. Geant4 - a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 5056(3), 2003.

[35] Cream CE. `http://grid.pd.infn.it/cream/`. Accessed at Apr. 2015.

[36] dCache. `https://www.gridpp.ac.uk/wiki/DCache`. Accessed at Apr. 2015.

[37] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *Communications of the ACM 51*, pages 107–113. ACM, 2008.

[38] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the Cloud: the montage example. In *SC IEEE/ACM*, page 50. IEEE/ACM, 2008.

[39] DIRAC, github repository. `https://github.com/DIRACGrid/DIRAC`. Accessed at Apr. 2015.

[40] DIRAC grid. `www.diracgrid.org`. Accessed at Apr. 2015.

[41] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992. in Italian.

[42] S. Easo, I. Belyaev, G. Corti, et al. LHCb simulation program. Technical report, LHCb, 2010.

[43] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative MapReduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818. ACM, 2010.

[44] J. Ekanayake, S. Pallickara, and G. Fox. MapReduce for data intensive scientific analyses. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, pages 277–284. IEEE, 2008.

[45] O. M. Elzeki, M. Z. Reshad, and M. A. Elsoud. Improved max-min algorithm in Cloud computing. *International Journal of Computer Applications*, 50(12):22–27, 2012.

[46] European Middleware Initiative. `http://www.eu-emi.eu/`. Accessed at Apr. 2015.

[47] Z. Fadika, E. Dede, M. Govindaraju, and L. Ramakrishnan. MARIANE: MapReduce implementation adapted for HPC environments. In *Grid Computing (GRID), 2011 12th IEEE/ACM International Conference on*, pages 82–89. IEEE, 2011.

[48] W. Fang, B. He, Q. Luo, and N. K. Govindaraju. Mars: Accelerating MapReduce with graphics processors. *Parallel and Distributed Systems, IEEE Transactions on*, 22(4):608–620, 2011.

[49] V. Fernández Albor, J. Cacheiro, R. Graciani, V. M. Muñoz, T. F. Pena, and J. Silva. User access to cvmfs software repositories on ibergrid. In *Iberian Grid Conference Ibergrid, Lisbon*. IOP Publishing, 2012.

[50] V. Fernández Albor, J. López-Cacheiro, F. Gómez-Folgar, R. Graciani, A. García-Loureiro, and J. Saborido. Study and performance of software scientific distribution over Cloud environments with cvmfs. In *XXII Jornadas de Paralelismo Universidad de La Laguna*, 2011.

[51] V. Fernández Albor and V. M. Muñoz. DIRAC MultiCloud Brokering. Technical report, Ibergrid, 2012.

[52] V. Fernández Albor, V. M. Muñoz, and T. F. Pena. BigDataDIRAC: deploying distributed Big Data applications. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, Guangdong, China*, 2015.

[53] V. Fernández Albor, V. M. Muñoz, and T. F. Pena. Federated big data for resource aggregation and load balancing with DIRAC. In *International Conference on Computational Science, Reykjavik, Iceland*, 2015.

[54] V. Fernández Albor, J. Saborido, F. Gómez Folgar, J. López Cacheiro, and R. Graciani Díaz. DIRAC integration with CloudStack. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 537–541. IEEE, 2011.

[55] V. Fernández Albor, M. Seco, V. M. Muñoz, R. Graciani, T. F. Pena, and J. Saborido. Multivariate analysis of variance for high energy physics software in virtualized environments. *International Symposium on Grids and Clouds, Academia Sinica, Taipei, Taiwan*, 2015.

[56] V. Fernández Albor, M. Seco, V. M. Muñoz, R. Graciani, T. F. Pena, and J. Silva. Cloud flexibility using DIRAC Interware. In *Journal of Physics: Conference Series*, volume 513, page 032031. IOP Publishing, 2014.

[57] T. Fifield, A. Carmona, A. Casajús, R. Graciani, and M. Sevior. Integration of Cloud, Grid and local cluster resources with DIRAC. In *Journal of Physics: Conference Series*, volume 331, page 062009. IOP Publishing, 2011.

[58] W. Ford et al. [RFC3280] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL). Technical report, The Internet Engineering Task Force, 2002.

[59] I. Foster. What is the Grid? a three point checklist. *GRIDtoday*, 1(6), 2002.

[60] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The physiology of the Grid: An open Grid services architecture for distributed systems integration. In *Open Grid Service Infrastructure Working Group, Global Grid Forum*, 2002.

[61] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International journal of High Performance Computing applications*, 15(3):200–222, 2001.

[62] I. Foster, Y. Zhao, R. Ioan, and L. Shiyong. Cloud computing and Grid computing 360-degree compared. In *Grid Computing Environments Workshop*, 2008.

[63] FUSE. `http://fuse.sourceforge.net`. Accessed at Apr. 2015.

[64] G. Corti, Monte Carlo simulation(s) in LHCb and introduction to Gauss. `https://lhcb-doc.web.cern.ch/lhcb-doc/presentations/TalksNotDirectlyRelatedLHCb/Postcript/GCorti-MCinHEP-2008.pdf`. Accessed at Apr. 2015.

[65] Z. W. Geem, J. H. Kim, and G. V. Loganathan. A new heuristic optimization algorithm: harmony search. *Simulation*, 76(2):60–68, 2001.

[66] gLite - Lightweight Middleware for Grid Computing. `http://grid-deployment.web.cern.ch/grid-deployment/glite-web`. Accessed at Apr. 2015.

[67] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[68] D. Goldberg and J. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99, 1988.

[69] F. Gomez-Folgar, A. García-Loureiro, and J. López-Cacheiro. An open-source Cloud management platform comparison. *5th Open Cirrus Summit*, 2011.

[70] Google App Engine. `http://code.google.com/appengine/`. Accessed at Apr. 2015.

[71] R. Graciani Díaz, A. Casajús Ramo, A. Carmona-Agüero, T. Fifield, and M. Sevior. BELLE-DIRAC setup for using Amazon Elastic Compute Cloud. *Journal of Grid Computing*, 9(1):65–79, 2011.

[72] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox. MapReduce in the Clouds for science. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 565–572. IEEE, 2010.

[73] S. Guo. *Hadoop Operations and Cluster Management Cookbook*. Packt Publishing, 2013.

[74] GZIP. `http://www.gzip.org`. Accessed at Apr. 2015.

[75] F. Gómez-Folgar, J. López-Cacheiro, C. Fernández Sanchez, A. García-Loureiro, R. Valín, and V. F. Albor. Formiga Cloud platform description. In *XXII Jornadas de Paralelismo Universidad de La Laguna*, 2011.

[76] Hadoop-on-demand, github repository. `https://github.com/alcachi/hadoop-on-demand`. Accessed at Apr. 2015.

[77] A. Harutyunyan et al. A practical approach to virtualization. In *The European Physical Journal Plus*, volume 126, pages 1–8, 2011.

[78] HEP-SPEC06. `https://twiki.cern.ch/twiki/bin/view/FIOgroup/TsiBenchHEPSPEC`. Accessed at Apr. 2015.

[79] K. Hinkelmann. Fisher. The design of experiments. In *John Wiley & Sons Inc.*, volume 1, 2007.

[80] C. Hong, D. Chen, W. Chen, W. Zheng, and H. Lin. MapCG: Writing parallel program portable between CPU and GPU. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pages 217–226. ACM, 2010.

[81] IaaS, PaaS, SaaS, Explained and Compared - apprenda.com. `http://apprenda.com/library/paas/iaas-paas-saas-explained-compared/`. Accessed at Apr. 2015.

[82] IN2P3. `http://www.in2p3.fr`. Accessed at Apr. 2015.

[83] A. Ishikawa et al. Evtgen: A montecarlo generator for b-physics. Technical report, Cern, 2004.

[84] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of 1995 IEEE International Conference on Neural Networks*, pages 1942–1948, Perth, Australia, 1995.

[85] G. Kousiouris, G. Vafiadis, and T. Varvarigou. A front-end, Hadoop-based data management service for efficient federated Clouds. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 511–516. IEEE, 2011.

[86] E. Lanciotti. An alternative model to distribute VO specific software to WLCG sites: a prototype at PIC based on CernVM File System. In *Cern,GDB meeting*, 2010.

[87] E. Lanciotti, J. Blomer, et al. An alternative model to distribute vo software to wlcg sites based on CernVM-fs: a prototype at pic tier1. Technical report, 2011.

[88] Y.-H. Lee, S. Leu, and R.-S. Chang. Improving job scheduling algorithms in a Grid environment. *Future generation computer systems*, 27(8):991–998, 2011.

[89] C.-C. Lin and D.-J. Deng. Dynamic load balancing in Cloud-based multimedia system using genetic algorithm. In *Advances in Intelligent Systems and Applications-Volume 1*, pages 461–470. Springer, 2013.

[90] Linux KVM. `http://www.linux-kvm.org/page`. Accessed at Apr. 2015.

[91] H. Liu and D. Orban. Cloud MapReduce: A MapReduce implementation on top of a Cloud operating system. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 464–474. IEEE Computer Society, 2011.

[92] S. Loughran, J. M. Alcaraz Calero, A. Farrell, J. Kirschnick, and J. Guijarro. Dynamic Cloud deployment of a MapReduce architecture. *Internet Computing, IEEE*, 16(6):40–50, 2012.

[93] M. Lu, L. Zhang, H. P. Huynh, Z. Ong, Y. Liang, B. He, R. S. M. Goh, and R. Huynh. Optimizing the MapReduce framework on Intel Xeon Phi coprocessor. In *Big Data, 2013 IEEE International Conference on*, pages 125–130. IEEE, 2013.

[94] X. Lu, F. Liang, B. Wang, L. Zha, and Z. Xu. DataMPI: extending MPI to Hadoop-like Big Data computing. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 829–838. IEEE, 2014.

[95] J. López Cacheiro et al. Leveraging EGI federated Cloud infrastructure for Hadoop analytics. In *Proc. of 8th Iberian Grid Infrastructure Conference, IBERGRID'14*, Aveiro, Portugal, Sept. 2014.

[96] G. Marliss and M. Menachem. Slaying the software dragon: A practical guide to software quality. In *International Thomson Computer Press*, 1997.

[97] A. Monaco. A view inside the Cloud. *The institute, The journal of IEEE news source*, 2012.

[98] V. M. Muñoz, A. Casajus, V. Fernández Albor, R. Graciani, and G. M. Arévalo. Rafhyc: An architecture for constructing resilient services on federated hybrid Clouds. In *Journal of Grid Computing*, volume 11, pages 753–770. IOP Publishing, 2013.

[99] V. M. Muñoz, A. Casajus, R. Graciani, and G. Merino. Use case: Running Montecarlo LHCb simulations using DIRAC with EGI federated Cloud. In *EGI Technical Forum Book of Abstracts*. EGI, 2012.

[100] V. M. Muñoz, V. Fernández Albor, R. Graciani Díaz, A. Casajús Ramo, T. F. Pena, G. Merino Arévalo, and J. J. Saborido Silva. The integration of CloudStack and OCCI/OpenNebula with DIRAC. In *Journal of Physics: Conference Series*, volume 396, page 032075. IOP Publishing, 2012.

[101] OGF DRMAA Working Group. `http://drmaa.org/documents.php`. Accessed at Apr. 2015.

[102] OpenAFS. `http://www.openafs.org`. Accessed at Apr. 2015.

[103] OpenNebula. `http://opennebula.org/`. Accessed at Apr. 2015.

[104] OpenStack. `http://www.openstack.org/`. Accessed at Apr. 2015.

[105] O. O'Malley. Terabyte sort on Apache Hadoop. `http://sortbenchmark.org/YahooHadoop.pdf`, May 2008. Accessed at Apr. 2015.

[106] M. Parastoo, B. Franck, B. Arne, et al. Migrating legacy applications to the service Cloud paradigm: The REMICS Project. In *European Research Activities in Cloud Computing*, pages 97–119. Cambridge Scholars Publishing, 2012.

[107] PIC. `http://www.pic.es`. Accessed at Apr. 2015.

[108] Proxy Globus. `http://dev.globus.org/wiki/Security/ProxyCertTypes`. Accessed at Apr. 2015.

[109] PyGrub. `https://wiki.debian.org/PyGrub`. Accessed at Apr. 2015.

[110] Pythia Collaboration. http://home.thep.lu.se/∼torbjorn/Pythia.html.

[111] P. Riteau, A. Iordache, and C. Morin. Resilin: elastic MapReduce for private and community Clouds. Technical Report RR-7767, Inria, France, Oct. 2011.

[112] Sahara - OpenStack. `https://wiki.openstack.org/wiki/Sahara`. Accessed at Apr. 2015.

[113] M. Salgueiro, P. González, T. F. Pena, and J. Cabaleiro. Assessment, design and implementation of a private Cloud for MapReduce applications. *Open Access Library Journal*, 1(3), 2014.

[114] Sample job config file. `http://wiki.apache.org/hadoop/JobConfFile`. Accessed at Apr. 2015.

[115] S. Sciutto. AIRES. A system for air shower simulations. Technical report, 2002.

[116] SPEC. `http://www.spec.org`. Accessed at Apr. 2015.

[117] P. Suri and M. Singh. An efficient decentralized load balancing algorithm for Grid. In *Advance Computing Conference (IACC), 2010 IEEE 2nd International*, pages 10–13. IEEE, 2010.

[118] J. Talbot, R. M. Yoo, and C. Kozyrakis. Phoenix++: Modular MapReduce for shared-memory systems. In *Proceedings of the second international workshop on MapReduce and its applications*, pages 9–16. ACM, 2011.

[119] C. Tang. A high-performance virtual machine image format for Cloud. In *USENIXATC Proceedings of the USENIX conference on USENIX annual technical conference*, pages 18–18, 2011.

[120] TAR. `http://www.gnu.org/software/tar/`. Accessed at Apr. 2015.

[121] The Large Hadron Collider. `http://lhc.web.cern.ch`. Accessed at Apr. 2015.

[122] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems*, 28(2):358–367, 2012.

[123] A. Tsaregorodtsev et al. Status of the DIRAC Project. In *Journal of Physics: Conference Series*, volume 396, page 032107. IOP Publishing, 2012.

[124] A. Tsaregorodtsev, V. Garonne, J. Closier, M. Frank, C. Gaspar, E. van Herwijnen, F. Loverre, S. Ponce, R. G. Diaz, D. Galli, et al. DIRAC–distributed infrastructure with remote agent control. In *Proc. of CHEP2003*, 2003.

[125] A. Turing. Intelligent machinery. Technical report, National Physical Laboratory, 1948.

[126] Twenty-One Experts Define Cloud Computing. `http://cloudcomputing.sys-con.com/node/612375/print`. Accessed at Apr. 2015.

[127] UNICORE. `http://www.unicore.eu/unicore/`. Accessed at Apr. 2015.

[128] Unified Modeling Language. `http://en.wikipedia.org/wiki/Unified_Modeling_Language`. Accessed at Apr. 2015.

[129] Universidade de Santiago de Compostela. `http://www.usc.es`. Accessed at Apr. 2015.

[130] V. Vapnik and S. Kotz. *Estimation of Dependences Based on Empirical Data*. Springer, 2006.

[131] VMDIRAC, github repository. `https://github.com/DIRACGrid/VMDIRAC`. Accessed at Apr. 2015.

[132] VMDIRAC OCCI. `https://github.com/DIRACGrid/VMDIRAC/wiki`. Accessed at Apr. 2015.

[133] VMWare ESX. `http://www.vmware.com`. Accessed at Apr. 2015.

[134] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen. G-Hadoop: MapReduce across distributed data centers for data-intensive computing. *Future Generation Computer Systems*, 29(3):739–750, 2013.

[135] Web Service Definition Language (WSDL). `http://www.w3.org/TR/wsdl`. Accessed at Apr. 2015.

[136] Windows Azure. `http://www.windowsazure.com`. Accessed at Apr. 2015.

[137] WLCG. `http://wlcg.web.cern.ch`. Accessed at Apr. 2015.

[138] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[139] XEN. `http://www.xen.org`. Accessed at Apr. 2015.

[140] X.-S. Yang and S. Deb. Cuckoo search via lévy flights. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 210–214. IEEE, 2009.

[141] C. S. Yeo, R. Buyya, M. Dias de Assunção, J. Yu, A. Sulistio, S. Venugopal, and M. Placek. Utility computing on global Grids, Grid computing and distributed systems (GRIDS). Technical report, 2010.

[142] L. Youseff, R. Wolski1, B. Gorda, and C. Krintz. Paravirtualization for HPC systems. In *In Frontiers of High Perf. Comput. and Networking - ISPA 2006 Workshops*, page 4431, 2006.

[143] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012.

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA