Elastic Grid Resources using Cloud Technologies

Joachim Christoffer Carlsen

Master's thesis in Software Engineering at

Department of Computing, Mathematics and Physics,
Bergen University College

Department of Informatics,
University of Bergen

June 2014

HØGSKOLEN I BERGEN

BERGEN UNIVERSITY COLLEGE

Acknowledgements

I would like to thank all those who contributed to this project. First, I would like to thank my supervisors Bjarte Kileng and Kristin Fanebust Hetland. Bjarte did much valuable work setting up the local AliEn systems and provided good feedback on the project progress. I would also like to thank Håvard Helstrup and Kristin for their feedback and advice concerning the writing of this thesis. Additionally, I would like to thank the AliEn staff for their feedback during the ALICE Offline meetings. I would also like to thank my study companion Ståle Nestås for his valuable work and help with the local AliEn site setup.

Finally, I would like to thank my family for their support, patience and understanding during my education as a whole and during this project.

Abstract

A Large Ion Collider Experiment (ALICE)[1] is one of four experiments at the Large Hadron Collider (LHC) at CERN[2]. The detectors in the ALICE experiment produce data at a rate of 4 GB/s after being filtered and compressed online. The data are stored and processed in a Grid system[3]. A Grid system allows for sharing globally distributed computing resources crossing administrative domains. The ALICE collaboration have created its own Grid middleware called Alice Environment (AliEn)[4] to facilitate the processing and storage.

This project will examine a possible way of better utilizing AliEn computing resources by using Cloud techniques[5], more specifically OpenStack[6] together with the virtual appliance CernVM[7]. Cloud techniques allow for adding and removing virtual computing resources through an API, providing elasticity in a computing center. This technique gives the possibility of removing the need for physical dedicated AliEn computer resources, and instead make them disposable; the virtual computing resources should only exist while needed.

This report will begin with a short general introduction and history of the technologies used in this thesis, followed by an introduction to Grid technology and AliEn. An introduction to Cloud technologies, OpenStack, and Virtual machines will then follow. After introducing the main concepts and tools, a description of a testbed and its setup process will be given, followed by an implementation of a prototype. Lastly, a short performance test, evaluation of the prototype and conclusions will follow.

Results show that implementing an elastic AliEn site using Cloud techniques is indeed feasible. The solution give an overhead of ~2:30 minutes per AliEn job agent, which is short compared to the lifespan of AliEn job agents, which normally is of 48 hours. Additionally, some possible ways of further reducing the overhead will be described in this report.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Background

1.1 Background

The ALICE (A Large Ion Collider Experiment) project is an experiment at CERN which aim is to study the physics of strongly interacting matter at extreme energy densities[1]. When these interactions occur, the collider's detectors record huge amounts of data which must be stored and processed; more than what a single computer or even a single data center can handle. For this purpose, Grid systems have been developed.

A Grid system is composed of geographically distributed computing resources facilitated by a Grid middleware. It is designed to run computing tasks that are too demanding for a single computer to handle, and to store huge amounts of data. Combined, these distributed computer resources make a "virtual super computer" which can be used by multiple organizations from all over the world[8]. A more detailed description of Grid systems will be given in chapter 3.

Cloud computing[5] is a different approach to distributed computing. Both Cloud and Grid systems offer large scale computing where the complexity of the software systems and the computer system is hidden for the end users. The differences of Grid and Cloud systems are found in the services provided by the systems. The Grid offer storage and an execution environment for jobs (typically an executable file with accompanying data to work with), while the Cloud provide predefined services which are divided into three groups:

- **Software as a Service (SaaS):** Typically web applications running in the Cloud, like Google Docs and ShareLatex.
- **Platform as a Service (Paas):** Typically a platform or container providing a sandbox environment for running software (often web-services) like Google

App Engine and Microsoft Azure.

- **Infrastructure as a Service (IaaS):** Most commonly associated with virtual machine hosting where the machine specifications can be defined by the customer (OS, RAM, Cpu-cores etc.), but usually also provide other virtual computing resources like networking and storage etc.

Cloud computing will be described in more detail in chapter 5.

1.2 Grid Middleware

A Grid Middleware is a set of software composed of components, services and protocols which automate machine to machine interactions, constituting a seamless distributed Grid accessible to its users[9]. The main purpose of a Grid middleware is to provide a secure envelope over all transactions, data management tools, single sign on, information services and APIs[3]. Some examples of Grid middlewares are Advanced Resource Connector (ARC), Open Science Grid, gLite, and AliEn, where the latter will be the one used in this project.

AliEn is a Grid middleware built upon existing open source components using a combination of web services and a distributed agent model. It started within the ALICE Off-line Project at CERN and constitutes the production environment for simulation, reconstruction, and analysis of physics data for ALICE[10].

1.3 Cloud Operating System

A Cloud operating system is a software system designed to control large pools of computer resources within a data center. Compute, storage and networking resources are managed through an interface like e.g. command line tools or a web interface, and are collectively used to provide virtual computing resources for customers, users, organizations etc. The main service provided by a Cloud OS is infrastructure (IaaS). From now on the term Cloud will be a reference to IaaS. A Cloud OS often consist of a hypervisor (a piece of computer software, firmware or hardware that creates and runs virtual machines) and resources such as a virtual machine disk image library, raw (block) and file-based storage, firewalls, load balancers, IP addresses, virtual local area networks (VLANs), and software bundles. Some examples of Cloud Operating Systems are Eucalyptus, CloudStack, Joyent, and lastly OpenStack which will be used in this project.

OpenStack[6] is an open source Cloud Operating System originally launched by RackSpace and NASA in 2010. Today the project consist of contributions from over 200 companies and over 9000 developers. The project is managed by the OpenStack Foundation, a non-profit corporate entity established in September 2012. OpenStack has a modular architecture that encompasses the following components (with code name): Compute (Nova), Object Storage (Swift), Image Service (Glance), OpenStack Identity (Keystone), Dashboard (Horizon), Networking (Quantum), Block Storage (Cinder). OpenStack APIs are to some degree compatible with Amazon EC2 (Elastic Compute Cloud) and Amazon S3 (Amazon Simple Storage Service) and thus client applications written for the Amazon Web Services can be used with OpenStack with minimal porting effort.

1.4 Motivation for this project

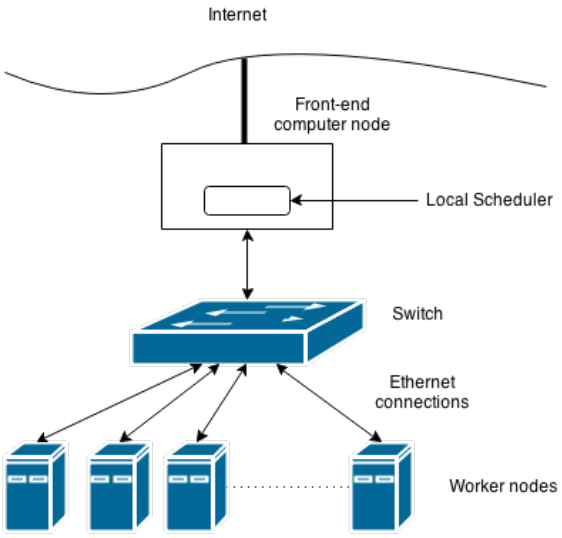

Figure 1.1: A local Grid cluster with physical dedicated resources

Today's AliEn Grid sites, and other typical Grid sites, uses dedicated resources for Grid middleware, as figure 1.1 show. A number of physical machines are dedicated to AliEn and its necessary software packages, where each machine can handle a given number of jobs simultanously, independent of the size of the jobs. When these resources are not running an AliEn job, they are still reserved for AliEn jobs instead of being free to use for other purposes. The motivation for this project is better utilization of these computer resources by using disposable virtual machines hosted on cloud systems. Cloud systems can ease addition and subtraction of physical

resources to Grid sites, providing elasticity within a data center, and better balance the load of virtual machines over the physical machines.

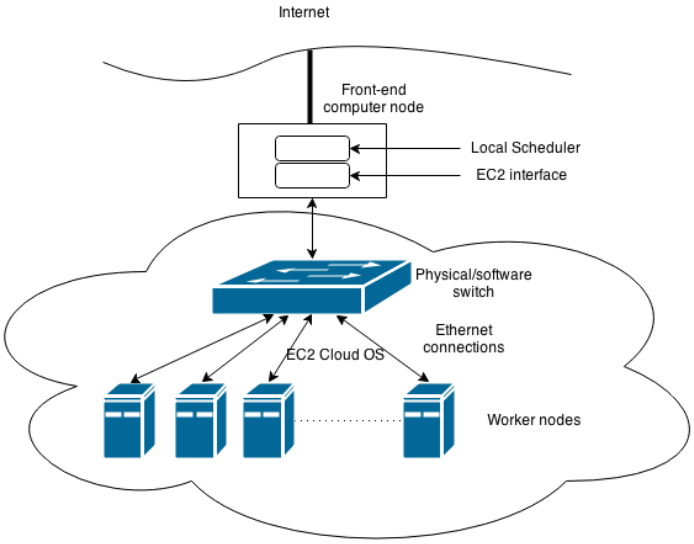

Figure 1.2: A local Grid cluster with dynamic resources using cloud techniques

This project will investigate the use of virtual appliances and Cloud techniques, CernVM[7] and OpenStack respectively, as a solution to this problem. By applying this solution to a Grid data center using the AliEn Grid middleware, virtual machines will be spawned on a data center when needed and be deleted when their work is done, automating their lifecycles, and additionally giving the possibility of limiting the maximum number of active virtual machines. Figure 1.2 display a sketch of the final implementation of the proposed solution.

The research method will be to build a prototype of a solution which functions and performance will be evaluated. Building the prototype will include developing a replacement of an AliEn component as well as a service for managing the lifecycle of virtual machines as Grid resources. After implementing these components, the potential loss of performance caused by virtualization will be measured, ending the thesis with an evaluation of the solution and the feasibility of applying the proposed solution on a Grid site.

Chapter 2

Background and history

In the introduction, a few new concepts and technologies were introduced. This chapter will give a short introduction to the history and applications of the technologies related to this thesis.

2.1 Computing in High Energy Physics

Computing is having an important role in todays science. The possibility of storing and processing immense amounts of data have revolutionized many fields. Allowing scientists to spend less time on automatable calculations and tasks, and instead analyze the results of these calculations, have been made possible by the introduction of computers and recently multi-core and -processor systems, and since the mid 90s, distributed computing.

One of the scientific fields heavily reliant on computing is the field of High Energy Physics (HEP). HEP adopted computing as a tool for storage and number crunching early in the 1970s and has been indispensable for the time after. The detectors within the ALICE experiment e.g. produce data at a rate of 4 GB/s after being filtered and compressed online[4].

The HEP field has not only been taking advantage of the computing technologies, but have also paved the way for many new technologies. One of the most notable is the World Wide Web (WWW) which was designed starting from the late 1980s by Tim Berners-Lee to help share data and discoveries among scientists.

2.2 CERN and ALICE

CERN (Conseil Européen pour la Recherche Nucléaire or European Organization for Nuclear Research) was founded in 1954 to establish a world-class physics research

organization and laboratory[2]. It is located on the Franco-Swiss border just outside Geneva, currently has 21 member states. 2400 people are employed at CERN and 10,000 particle physicists from 113 countries, about half of the worlds particle physicists, are participating in different experiments.

There are several ongoing projects at CERN studying everything from the biological effects of antiprotons[11] to how crystals could help to steer particle beams in high-energy colliders[12]. One of these projects, which is related to this thesis, is ALICE.

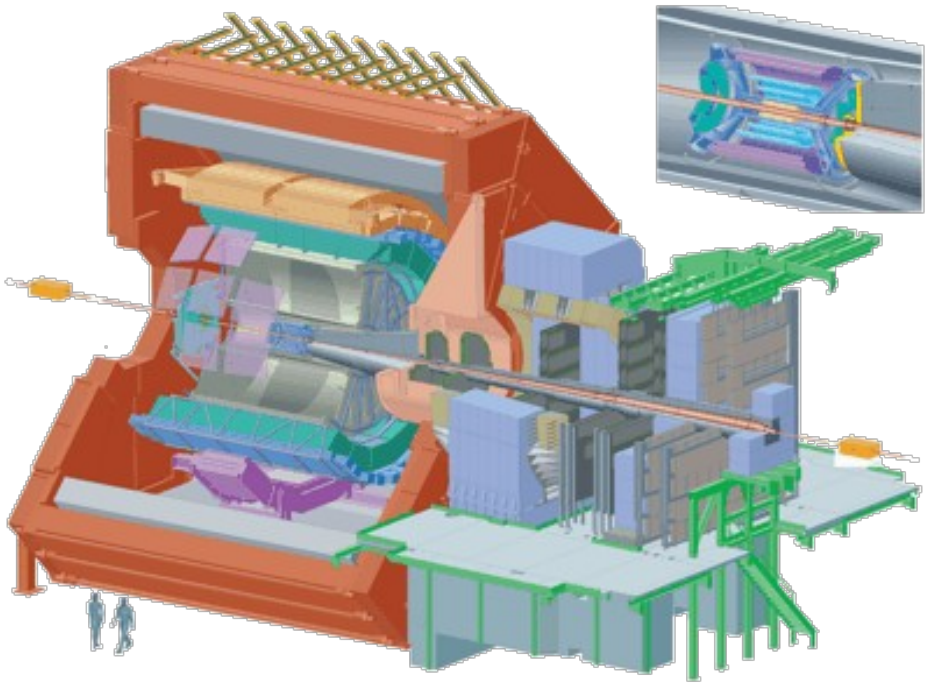

Figure 2.1: The ALICE experiment[1].

ALICE, A Large Ion Collider Experiment[1], is a heavy-ion detector (Figure 2.1) on the Large Hadron Collider (LHC) ring, designed to study the physics of strongly interacting matter at extreme energy densities, where a phase of matter called quark-gluon plasma is formed[13]. When interactions occur, the detectors in ALICE generate huge amounts of data. The AliEn Grid middleware was developed to help store, process and give access to these data.

2.3 Grid

The first types of distributed computing appeared in the 1960s and 1970s. In the beginning, these systems were mostly locally connected computers or multiprocessor

systems built for high performance computing. The first geographically distributed computing system was ARPANET (Advanced Research Projects Agency Network). It was deployed in 1969 and had initially a 50 Kbit/s network connecting four computer nodes in the United States.

Five years later, the Transmission Control Protocol (TCP) was introduced and four years further on the Internet Protocol (IP) was developed. IP was used together with TCP forming what is called TCP/IP, which quickly became universally adopted, and is still the backbone of todays internet[3].

The introduction of the globally distributed packet-switched network gave the possibility of geographically distributed computing, but implementing it was expensive and computer hardware and operating systems (OS) were also expensive. During the 1980s and 90s the hardware and OS cost decreased, and with accessibility to cheaper hardware and operating systems the first types of networked clusters, called Beowulf clusters[14], appeared. Named after a NASA project, the term was used for clusters composed of commodity hardware running the Linux operating system and connected by commodity ethernet switches. This solution proved to be more cost effective than using state of the art computers connected by specialized high speed cluster interconnections. The Beowulf clusters were among the first clusters looking like what today could be a Grid site.

On the 1995 Super Computer conference, the first large-scale Grid was demonstrated, named the I-Way project[3]. It included 17 geographically distributed supercomputer sites, and over 60 applications from various fields were demonstrated. This project was the start of the Globus Toolkit, a toolkit consisting of Grid middleware components such as security, job submission and resource management. This toolkit is still actively developed and many of its components are used in other Grid middleware projects, AliEn mentioned in 2.2 being one of them.

2.4 Virtual Machines

The first Virtual Machines were created in the 1960's when IBM wanted to share physical hardware between different users. The demand for computer access were increasing but the cost was high and the existing Operating Systems support for multiple concurrent users were limited. The virtual machines gave each user its own environment seemingly running directly on the hardware[15].

Later in the 60's virtual execution environment for intermediate languages were introduced. O-code[16] was the first intermediate language, produced by a Basic Combined Programming Language (BCPL)[17] compiler, and executed much like

todays java byte-code on the Java Virtual Machine (JVM)[18].

As the price of hardware decreased virtual machines served less purpose than earlier, resulting in lower interest in virtual machines, and the advances in the development of the virtual machines in the 80's and early 90's were few. People could now afford their own computers and the hardware produced for desktop computers had no support for virtualization.

Later in the 90's researchers and businesses saw the potential of virtual machines as a way of hosting several operating system instances in on one machine as well as to conserve server space[15]. When researchers at Stanford university found a way to virtualize on commodity hardware, the interest for virtual machines started growing again[19]. Today multiple virtual machine managers(VMM) exists. This software hosting virtual machines, each with different requirements and performance are trying to provide quicker virtualization than the other, driving the effectivity of virtualization forward.

2.5 Cloud Computing

Over the next decade after the internet was introduced, more and more companies, institutions and other customers wanted and needed to get online. Server hardware was getting increasingly expensive following the growth of demand, getting too expensive for smaller companies. To make servers affordable for smaller companies, the same techniques as IBM used in the 1960s was applied to split the servers into multiple machines. Servers could be virtualized into shared hosting environments, Virtual Private Servers and Virtual Dedicated Servers.

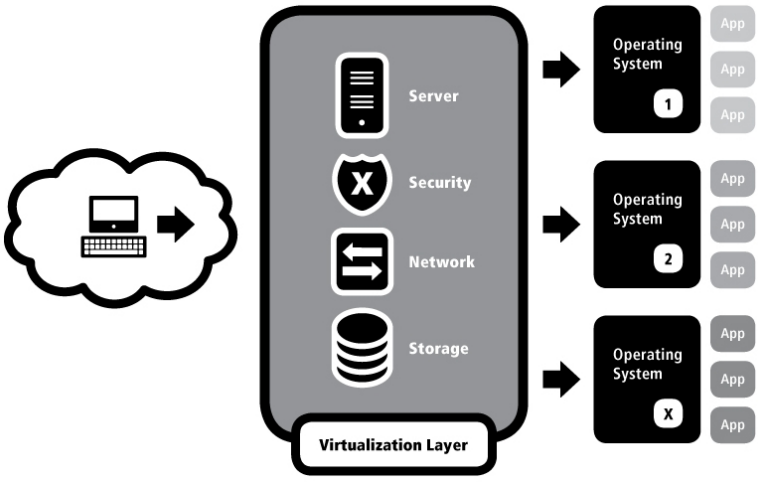

Figure 2.2: Virtualization on cloud resources is used to provide virtual hardware resources[20].

Later, with decreasing server hardware prices, more people and companies could afford their own servers and also building server parks. With an increase of internet consumers, sites needed bigger servers and server parks to handle the increase of traffic. There was now a need to combine resources as shown in figure 2.2. This was solved by using what today is called Cloud Computing or Utility Computing. With hypervisors (VMMs) installed on multiple nodes, the combined computing resources could be represented as one node with the combined power of all the nodes.

Today a combination of virtualization and combined computers are popular in public Infrastructure as a Service clouds. Data centers are connected by a Cloud Operating System combining its resources. These resources are split into different scalable sizes to provide multiple scalable virtual machines for customers[20].

Chapter 3

Grid computing

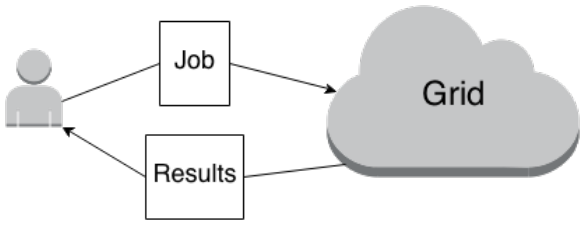

Figure 3.1: A user send a job to the Grid without being concerned about where the job is executed.

A simplified view of the Grid as shown in figure 3.1 is something which a user or service can send a job (see Section 3.4.1) to and retrieve the results of that job later. Since its introduction Grid systems have become one of the most important tools enabling processing of large amounts of data requiring high processing power, and large amounts of memory and input/output operations. These processing demands especially applies to the scientific world of high energy physics, but also in other technological, engineering and business areas. This chapter will give an introduction to some of the key aspects of Grid computing.

3.1 Introduction

Grid computing uses geographically distributed interconnected computer resources collectively to achieve higher performance computing and resource sharing. It was first displayed in the mid-1990s, and the growth of high-speed networks and the Internet allowed distributed computer systems to be easily interconnected[3].

Grid systems are often heterogenous both in terms of hardware and software, as a result of its distributed nature and decentralized control. Figure 3.2 shows different sites with different configurations on the same Grid, communicating using the public

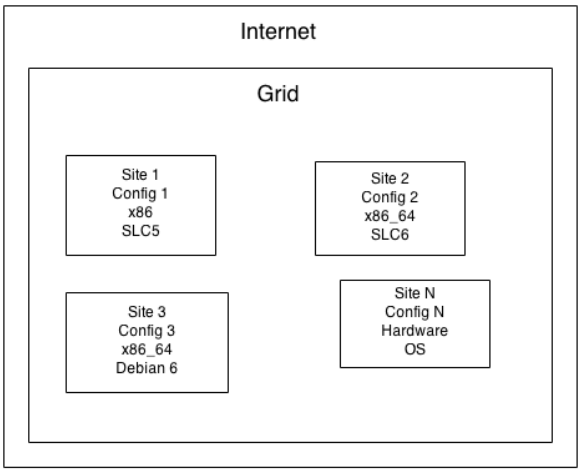

Figure 3.2: A Grid typically have sites with different hardware running different operating systems with different configurations.

internet. Ensuring safe data transport and access control over the Internet as well as effectively utilizing the resources are the main challenges of constructing a Grid middleware.

3.2 Virtual Organizations

Grid systems are normally used by between many different users. A virtual organization (VO) is a term used to describe an agreement between organizationally and geograhically distributed groups of organizations/institutes. These groups are working together for a common purpose, sharing their resources such as data, computers, databases, networks.

The purpose of a VO is to help managing a Grid by having the administrative responsibilities in a virtual domain, crossing multiple administrative domains. Resources can be shared between multiple Virtual Organizations, and to facilitate this resource sharing different Grid systems have been developed. The key aspect of a VO is its formation for a common purpose.

3.3 Security

In Grid computing it is important to provide secure data storage and transfer, to have access control to the resources, and to trust the different participants. There are many threats against data traffic, and as Grid systems are distributed, cross-domain and use public networks, they are also prone to these threats. Security techniques have been developed to protect data against eavesdropping and to secure that data

cannot be altered while under transportation. The terms *data confidentiality* and *data integrity* are used to describe this protection.

In a Grid system it is also important to ensure that an identity can be validated and that the provided identity have access to the requested resources. The terms used for this is *authentication* and *authorization*. These four terms constitute the basis for the Grid Security Infrastructure (GSI).

Public Key Infrastructure (PKI)[21] is a powerful arrangement that incorporate all the terms described above, ensuring a secure connection. PKI uses asymmetric key cryptography which is designed such that a user get two keys where one can decrypt messages encrypted by the other and vice versa, and both keys are associated with the owner. The keys are called *Public Key* and *Private Key* where the latter is kept private by the user, hence it's name.

Data integrity is ensured by digesting (encrypted hash of the original message) the message before sending and after receiving it. The digest is sent along with the message and compared to the digest of the recipient. If the digests differ, the message has been altered.

Data confidentiality is achieved by encrypting the data using the senders private key. The recipient may decrypt the message using the senders public key.

When authenticated, the user must be authorized for using a resource. In some grid middlewares, the existence of a user on the resource is required. On other the subject of a cetificate is stored together with the username of the certificate owner in a "grid-mapfile" giving the user access to the resource.

To ensure that a key originate from a trusted user, the certificates are signed by a Certificate Authority (CA). The CA is the ultimately trusted participant of a PKI, and have the responsibility of signing, distributing and revoking digital certificates. Revoked certificates are published as a list. This list contain certificates with valid dates, but which should not be accepted.

3.3.1 Proxy certificates

When interacting with a remote service, a certificate is used to authenticate the user. In a Grid system, these remote services often interact with other remote services on the user's behalf e.g. to transfer files, execute processes. To act on the user's behalf, the first remote service must prove that it is a delegation of the user.

A way of implementing delegation is to use proxy certificates, usually called a proxy. A proxy keeps its own public and private keys, and is signed by the certificate of the user which created it. It has a short lifecycle, and is usually valid for no more

than 12-24 hours.

3.4 Job Distribution

The term *job* in terms of Grid describes a program to be executed on the Grid initiated by a user. These programs can be software written by the user itself in languages like Java, C++, Python etc., or it can be a pre-compiled application package. This section will describe how these jobs typically are distributed in Grid systems.

3.4.1 Job definition

The programs mentioned above might have different parameters, i.e. which files to read data from and which files to write the result data to. They may also have requirements in terms of execution location, extra software packages etc. Together these parameters and requirements define a Grid job.

Jobs are defined differently on different Grid middlewares. ARC and Gridway use the Job Submission Description Language (JSDL)[22] which is an XML specification developed at the Global Grid Forum. AliEn uses the Job Description Language (JDL)[23] which is based on the Classified Advertisement (ClassAd)[24] language.

```
Type = "Job";
JobType = "Normal";
Executable = "myexe";
StdInput = "myinput.txt";
StdOutput = "message.txt";
StdError = "error.txt";
```

Table 3.1: Sample JDL file contents

A JDL file consist of lines on the following format: **attribute = expression;**. These attributes and expressions are used to describe job properties such as input and output files, executable files and other dependencies like the example in table 3.1. Computer resource requirements can also be specified. The job might require a minimum amount of memory, or access to a number of CPU cores to be executed.

3.4.2 File transfer

As seen in table 3.1 jobs may have input and output files, also with the possibility of multiple additional files. These files may be downloaded from a storage element somewhere on the Grid. For this purpose the FTP protocol is used with some

extensions giving it the name GridFTP. One of these extensions is giving support for enhanced security by applying GSI to the FTP protocol. Another is allowing increased download speed by enabling downloading different file segments from multiple sources. In addition to these two advantages, GridFTP also have increased fault tolerance and support for partial file transfers.

3.4.3 Load balancing

Balancing the jobs on a Grid is crucial to maximally utilize its effectiveness. There are several challenges to load balancing on the Grid contrary to super computers, and also regular PCs. The homogenous environments might have different performance for different jobs, the location of the job data may impact transfer times etc. Mechanisms for handling these challenges to properly balance load is necessary.

Load balancing on the Grid is a matter of having an overview of available/unavailable resources and queued jobs. Normally a broker service select the computer resource on which the job can be executed. There might be several criterias for a job which have to match with a computer resource to label it as a candidate computer resource. These criterias are normally listed in the job description file. If a candidate resource is available, the job will be sent to it and be executed there.

The method for distributing the jobs may vary from Grid to Grid. One such method is using *agents*. Agents are representatives of local grid resources and handle load balancing and resource discovery by communicating with each other[25]. This technique makes the information about a site and its computer resources easily available on the Grid system, alleviating the information gathering on the job brokering services.

3.4.4 Parallelization

A distributed Grid is well suited for performing parallel computing tasks. Parallelization is a term used to describe the technique of splitting the work of a program into smaller independent tasks, instead of executing all tasks sequentially. However, parallelization on the Grid have some prerequisites to perform optimally.

Parallel programs often need to communicate intermediate results between processes. With the possibility of these processes running on the opposite side of the earth, the cost of communication increase in terms of execution time. Therefore, parallel Grid jobs are often of the type coined "embarassingly parallel"; processes requiring minimal to no communication between themselves. A typical job in High Energy Physics is easily solveable. Data from an event in a detector usually needs

smaller amount of computing power to be analyzed, but as the events are numerous, the jobs are easily parallelized.

Chapter 4

Alice Environment (AliEn)

In the previous chapter, Grid systems and the main problems solved by Grid middleware were introduced. This chapter will in greater detail descibe the middleware used for this project, called AliEn (short for ALICE Environment). This chapter will give an introduction to its current status and describe the components of this Grid middleware.

4.1 Status

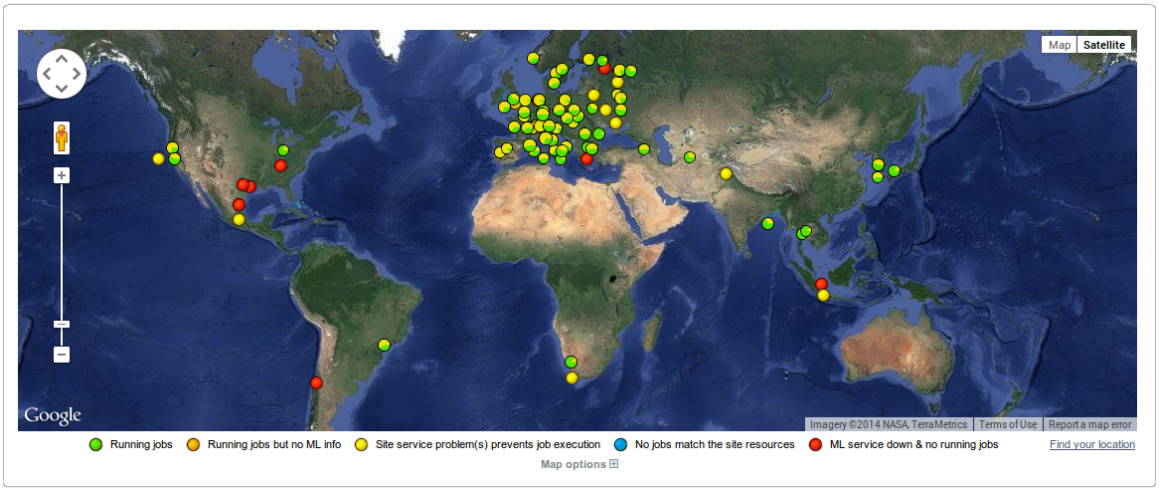

Figure 4.1: Display of the status of the AliEn Grid sites (including planned sites) as of May 7th 2014[26].

AliEn is currently running on 67 sites across the world and the number of sites are still growing. 53 sites are located in Europe, 8 in Asia, 4 in North America and one in both Africa and South America. In addition, 8 new sites are planned for future AliEn deployment. In total these sites execute on average 250 000 jobs per

day with an average job efficiency of 75 percent[27]. Figure 4.1 display the global distribution of the AliEn sites with their current status as of May 7th 2014.

4.2 Components

Systems for distributing jobs and receiving them, as well as monitoring the jobs and other components are needed for a Grid to function as a distributed system. The AliEn Grid middleware is composed of several components working independently. This section will describe these components and their responsibilities.

4.2.1 Job Execution

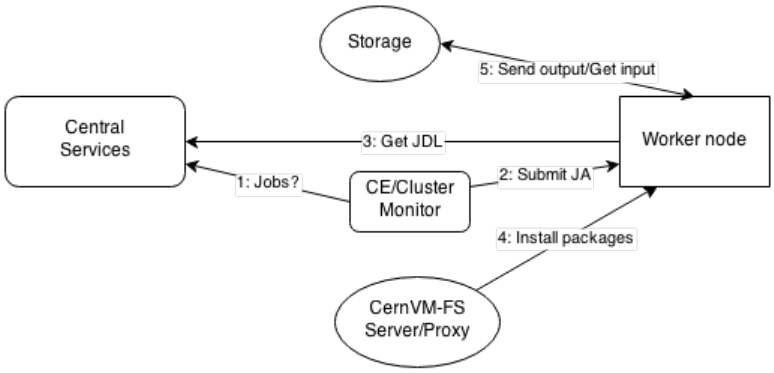

Figure 4.2: The different components involved in AliEn job execution, and the communication between them[28].

There are 5 actors in the job execution process displayed in figure 4.2. The Cluster Monitor is a service on each site keeping track of the Job Agents on the site. This service is also communicating with the Broker Service on the Central Services. The broker is matching job requirements with the information provided by the cluster monitors to find sites suitable to execute a job. A job is defined in the Job Description Language (JDL) and is stored in the database on the Central Services. The Compute Element (CE) is a service on each site responsible for executing jobs. They are usually pointing to a resource manager like TORQUE[29] or HTCondor[30]. When a job is matched with a site, the compute element create a Job Agent (JA) which is started on a computing resource or a Worker Node (WN). The JA is functioning as a wrapper for a job, doing any necessary preparations for a job to run, checking that the resource is capable of running the job, clean up after the job has finished. When the JA is started, it pulls the job from the Central Services and executes it.

4.2.2 Storage

As mentioned earlier, Grids are not only used for job execution, but also data storage. The large amounts of data read from the ALICE detector are stored on the sites composing the Grid. These sites usually have one or more storage elements (SE), although some have none. Most SEs use a highly scalable file server called Xrootd[31]. This file server provide a POSIX-like file access interface and is optimized for minimal latency and network efficiency.

4.2.3 Catalogue

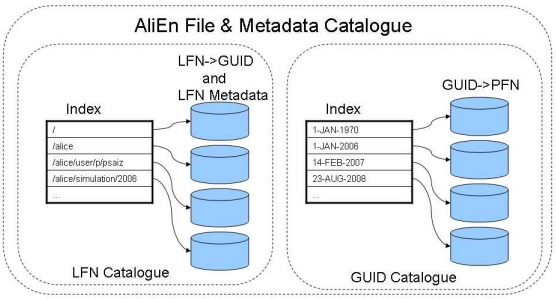

Figure 4.3: The AliEn file catalogue are navigated like a UNIX file system, although files may be on different locations. Behind the scenes the files are mapped to their physical location[32].

As displayed by figure 4.3, the AliEn file and metadata catalogue is a database with pointers, or Logical File Names (LFN), to the files physical locations (PFN), provoding mapping of files between diffferent SEs on different sites. From the perspective of a user, the catalogue are navigated similarly to a UNIX filesystem.

4.2.4 Monitoring

MonALISA (Monitoring Agents using a Large Integrated Services Architecture)[33] is the monitoring service employed by AliEn. It is collecting information on the systems as well as working on optimization of workflows. Additionally it provides a web GUI visualisin the Grid as displayed in figure 4.1.

4.3 Installation of packages

When the job agent prepare for the execution of a job, it is also downloading the software packages needed by the job. Over the years of AliEns existence, three different techniques have been used for this purpose, each developed to replace the others. All three are still being used, but most sites have transitioned to use the newest.

The first installer is PackMan, a package manager for homogenous environments. It keeps a list of installed and available packages, as well as installing, uninstalling them, and handling dependencies.

The second is the torrent installer. As the name implies, it is using the torrent protocol to download software packages. This technique downloads and install AliEn using torrent, and then install the packages. Over the last few years it has been replaced with CernVM-FS.

CernVM-FS (CVMFS), as described in section 6.4.2, is the last and newest of the installation methods, and is being transitioned to today. This technique give access to both AliEn and the required packages as if they were already installed on the system.

Chapter 5

Cloud technologies

The last two chapters gave an introduction to the first core technology of this project. This chapter will focus on the second which is Cloud technologies, more specifically Infrastructure as a Service, a technology taking advantage of virtualization (will be described in chapter 6) and related technologies.

5.1 Introduction

The US National Institute of Standards and Technology have developed the following definition for the term *Cloud Computing*.

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."[34]

The model is composed of five essential characteristics, three service models which was mentioned in section 1.1, and four deployment models.

Characteristics

- *On-demand self service:* Customers can administrate resource without human interaction with the service provider.
- *Broad network access:* Resource administration is available via standard communication platforms.
- *Resource pooling:* Resources are shared among multiple customers and a consumer have no control or knowledge over resource location.

- *Rapid elasticity:* The amount of resources used can be scaled on-demand at any time.
- *Measured service:* Resource usage can be monitored, controlled and reported.

Deployment models

- *Private cloud:* Infrastructure provisioned for use by a single organisation.
- *Community cloud:* Infrastructure provisioned for use by a specific community from organisations.
- *Public cloud:* Cloud infrastructure procisioned for use by the general public.
- *Hybrid cloud:* A hybrid of the above deployment models.

Clouds are considered a solution to some of the problems encountered with early adaptations of Grid computing where the site retains control over the resources and the user must adapt their application to the local operating system, software and policies[35].

5.2 Why Grid instead of Cloud?

Both the Grid and the Cloud provides distributed computing, but in different forms. What makes the one better suited for scientific purposes than the other? Why is Grid used for this purpose instead of Cloud which is more widely used today for distributed computing?

The main reason is that the Cloud does not offer the execution environment which the Grid does. Platform as a Service (PaaS) is the closest the cloud get to provide what is needed for scientific purposes, but PaaS typically offer a container for web applications/services. This container is often a sandbox envirnoment limited to one specific programming language and with limitations concerning network, file access and software libraries. There are typically no Cloud service offering an environment for executing arbitrary executables, compiled or not, written in the user's preferred language.

What the Grid need is computer infrastructure, which can be provided virtually by one of the three Cloud service models, Infrastructure as a Service. How the Grid might benefit of this model will be discussed in the next section (5.3).

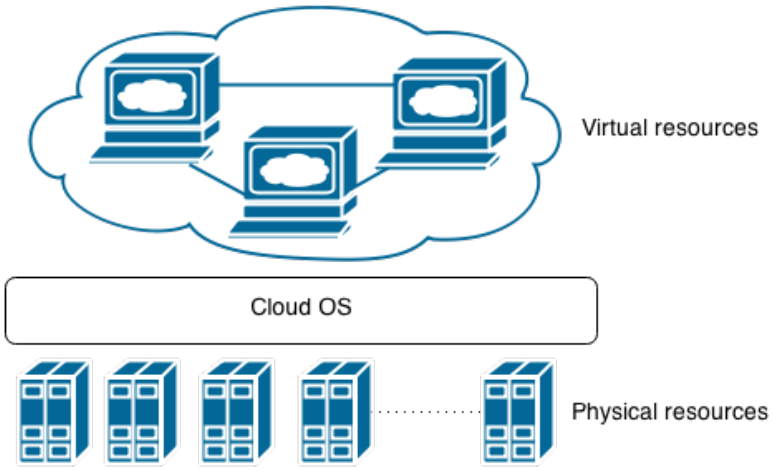

Figure 5.1: The Cloud operating system use physical computer resources to provide virtual scalable virtual resources like virtual machines and virtual networking.

5.3 Infrastructure as a Service

Infrastructure as a Service (Figure 5.1) and the systems providing this service will from now on be referred to as Cloud Operating Systems or Cloud OS. A Cloud OS provides computing resources like virtual machines, storage, networking (bandwidth, dhcp, load balancing) etc. for customers, users, organisations etc. This model has several benefits like scalability, location independence, physical security of data centre locations and no single point of failure. Most of these are most interesting from a business perspective. The most interesting feature of IaaS concerning this project is the scalability provided by IaaS.

As mentioned in the introduction (Section 1.4), this thesis aims for elastic Grid computing resources. These resources should be created on demand and deleted when the resources are no longer needed. This can be achieved by taking advantage of the scalability provided by IaaS. By creating an interface between a Cloud OS and a Grid middleware, Grid resources can be spawned and deleted based on the current need for computer resources.

5.4 Elastic Compute Cloud (EC2)

5.4.1 Introduction to EC2

EC2 is the Insfrastructure as a Service component of Amazon's public cloud platform. It was officially released in October 2008 after being in beta since August 2006, and has since its release gained a huge share in the public Cloud IaaS market. Figure 5.2

give some insight to the magnitude of Amazon's market share compared to its main competitors on the IaaS and PaaS market.

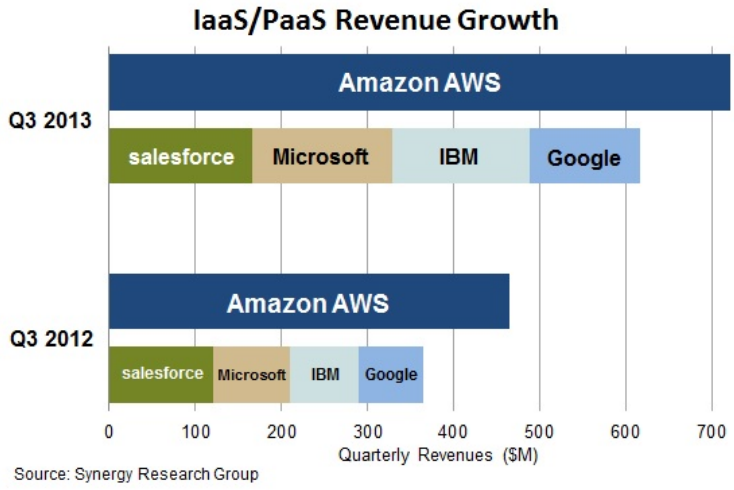

Figure 5.2: Cloud revenue comparison[36]. It is clear that Amazon IaaS/PaaS have the larger market share.

5.4.2 EC2 API

Different ways to interact with IaaS systems have been developed. Web interfaces, command line tools and APIs have been developed for different Cloud systems to provide various ways of interaction. Among these are the EC2 Application Programming Interface (API). The EC2 API is designed to send and receive commands or data to Amazon Web Services (AWS), primarily to virtual machines in the EC2 system, but it has also over the years been adopted by other IaaS systems.

5.4.3 Why use the EC2 API?

Due to EC2s great popularity several cloud system developers have adopted the EC2 API to their IaaS projects to give users a smoother transition from the public cloud to a private/hybrid cloud solution. In addition to being widely used by other cloud systems, it also provide our project with the necessary functionality for starting, stopping, communication and transmitting user data etc. to the virtual machines.

Instead of using a cloud operating system's specific API (in our case OpenStack's own API), EC2's API is more general and is used alongside several other cloud OS APIs like OpenNebula[37], Apache Cloud Stack[38] and Eucalyptus[39]. This allow our project to be run together with other cloud OSes than OpenStack giving eventual users (datacenters) more options when choosing Cloud OS.

5.5 OpenStack

Infrastructure as a Service has been introduced along with the selected API to access such a service. The IaaS system chosen for this project is OpenStack[6]. The reason behind this selection is based on the large user and developer base of OpenStack.

5.5.1 Introduction to OpenStack

OpenStack is an IaaS software stack composing a cloud operating system originally developed by NASA[40] and RackSpace[41]. It is released under the Apache Licence, making it free and open source. Its mission is to "produce the ubiquitous Open Source Cloud Computing platform that will meet the needs of public and private clouds regardless of size, by being simple to implement and massively scalable[42]." Along with its growth many companies have joined the development adding up to more than 200 companies by the end of 2013, including big names like AT&T, IBM, Intel, Oracle and VMware.

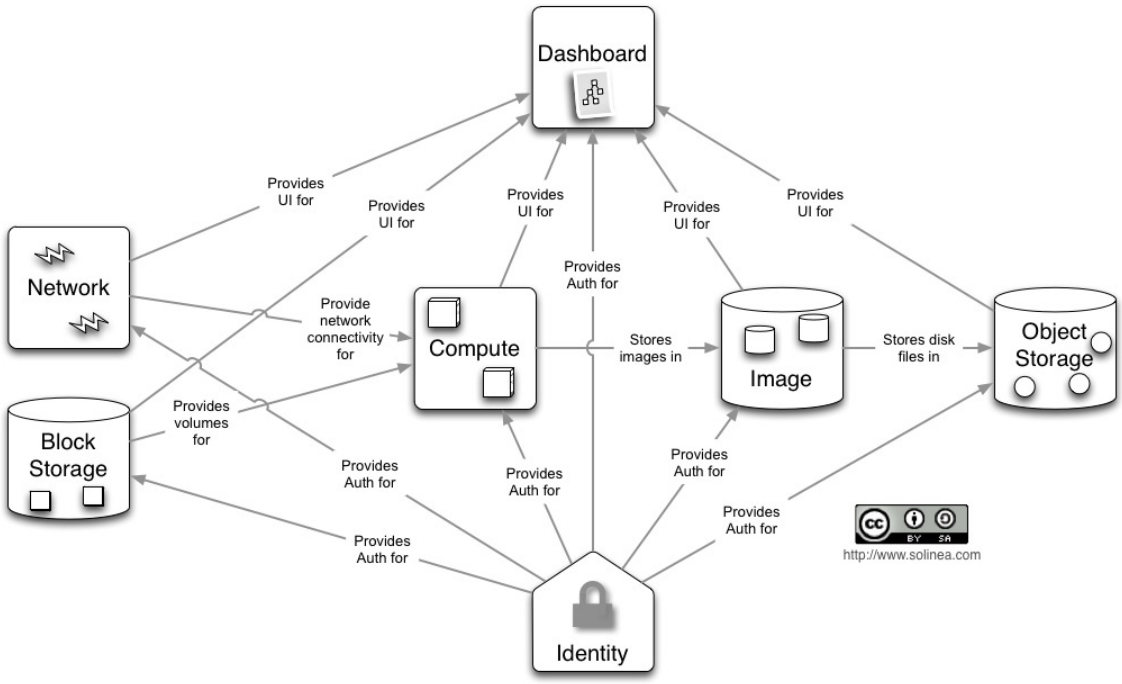

Figure 5.3: The different components of OpenStack and how they are related to each other[43].

OpenStack has a modular architecture with various components serving different purposes, allowing to spread functions of the systems over multiple computer resources. Figure 5.3 show the different components of OpenStack and how they are related to each other. The components main responsibilities, along with the names

of the components, are[43]:

- **Compute (Nova)** providing the execution platform for the virtual machines.
- **Object Storage (Swift)** a general storage service used to store e.g. images.
- **Block Storage (Cinder)** providing virtual storage devices for virtual machines.
- **Networking (Neutron)** a networking service giving IP addresses to virtual machines and managing the virtual networks.
- **Dashboard (Horizon)** a web interface to administrate organisations/users virtual machines.
- **Identity Service (Keystone)** the authorization and authentication service of OpenStack.
- **Image Service (Glance)** for storing virtual machine images.

This project will mostly have to interact with Nova, the Compute service, but also during the setup process Keystone will be used for creating user accounts and EC2 credentials, Glance for OS image handling and Cinder for creating and using volumes.

5.6 OpenStack concepts

Along with the different components of OpenStack is a set of concepts used among these components. This section will describe some of the OpenStack concepts relevant for this thesis.

5.6.1 Users and Tenants

OpenStack is designed to be used by different consumers and groups of consumers. A tenant is a representation of a project and its resources quotas. It has a defined limits for resources such as RAM, instances, volumes, Virtual CPUs, and IP ranges.

A user is a participant of one or more tenants with different privileges on each tenant. It may have different roles with different access, like assigning public IP addresses, starting stopping instances.

The user/tenant model is well suited for our project. A site running OpenStack may set up a tenant for different VOs having their own set of rules, users and resources.

5.6.2 Instances

Instances in OpenStack are individual VM instances running on physical compute nodes. An instance must have a few properties specified before started, like which volumes to attach, which flavor, security groups, networks etc. These properties are a combination of different concepts which will be described below.

5.6.3 Volumes

OpenStack offers the possibility to create predefined disks which can be attached to virtual machine instances using the Logical Volume Manager on Linux. Some Virtual Machine Appliances (see section 6.4) are distributed as disk partitions, e.g. CernVM (see section 6.4.1). As CernVM (CMV) is a partition containing a Linux installation, a CVM partition can be added as a volume to OpenStack. This way a virtual machine instance can attach the volume and boot up from it.

5.6.4 Virtual Machine Flavors

A VM flavor is a set of predefined virtual hardware configurations that can be selected when booting a VM instance. By default there are five flavors distributed along with OpenStack as listed in table 5.1.

ID	Name	Memory (MB)	Disk (GB)	Ephemeral (GB)	VCPUs
1	m1.tiny	512	1	0	1
2	m1.small	2048	10	20	1
3	m1.medium	4098	10	40	2
4	m1.large	8192	10	80	4
5	m1.xlarge	16384	10	160	8

Table 5.1: OpenStack is distributed with six different VM flavors.

Each flavor have a name and a unique ID in case of naming conflicts. Memory is specified in megabytes and disks in gigabytes. The Disk field is the size of the root disk of the virtual machine (/dev/vda) while the Ephemral field is an additional disk (/dev/vdb) which can be attached to the VM. The VCPU field correspond to the number of Virtual CPUs the VM can make use of. A few other parameters can also be specified, e.g. whether the flavor is public, swap size. For this project another flavor will be created to better suit the testing environment.

5.6.5 Security Groups

A security group is a set of IP and port filter rules. These rules are applied to the VM instances networking. When specifying a new rule for a security group, the port to be opened must be provided along with a range of IP-addresses which are allowed through the port.

5.6.6 EC2 User Data

EC2 User data is a file assosiated with a VM instance. The VM instance can download this file from a special URL (http://169.254.169.254/latest/user-data). The contents of the user data file is specified before starting a VM instance, and often contains different configuration data of different forms for the instance and/or executable scripts.

5.7 OpenStack with EC2

Although most EC2 functions are implemented in OpenStack, some functionality which could have been useful for this project have been removed or have not yet been implemented. One of these give the possibility of determining what to do when a machine is shut down. When creating a new virtual machine with EC2, an optional flag *shutdown_behavior* could be specified. An option for this flag is *terminate*. This would allow us to have the machine automatically deleted from the system once it got turned off. This functionality have unfortunately been removed because OpenStack could no longer see the need for this function[44].

Chapter 6

Virtual Machines

Virtualization and Virtual Machines are key components of Cloud Systems providing Infrastucture as a Service. This chapter will give an introduction to different virtualization techniques, ways to utilize virtualization and how virtualization is facilitating IaaS.

6.1 Introduction

A Virtual Machine (VM) is a software implementation of a computer, and is executing programs like on a traditional physical machine. This software is executed on top of a Virtual Machine Manager (VMM), often called a hypervisor. Virtual machines are separated into two classes; System virtual machines and process virtual machines.

A process virtual machine, often called Managed Runtime Environment, is an execution platform for a single process. These virtual machines are usually run as a normal application on its host OS. An example is the Java Virtual Machine providing an abstraction layer for a range of programming languages, most important Java, but also other languages are gaining popularity as they're maturing, like Scala, Clojure and Groovy.

A System virtual machine provides a complete system platform allowing the execution of complete operating systems. This gives the possibility of hosting multiple isolated operating systems on the same hardware (i.e. computer). Kernel Virtual Machine (KVM), VMware and VirtualBox are some examples of system virtual machine software. This class is the most interesting for this project. The term VM will from now refer to System VM.

Virtual machines introduce overhead impacting the performance of the guest OS and its applications. Although with hardware virtualization support, proper hardware and setup, the performance loss can be greatly reduced.

6.2 Virtualization

Today the term virtualization is usually associated with x86 virtualization. The term x86 is used for instruction set architectures which are backward compatible and based on the Intel 8086 CPU from 1978. Lots of instructions have later been added to the set of which most are backward compatible. Among the added instructions are the support for hardware virtualization. Before hardware virtualization was added, other approaches for achieving virtualization were developed instead.

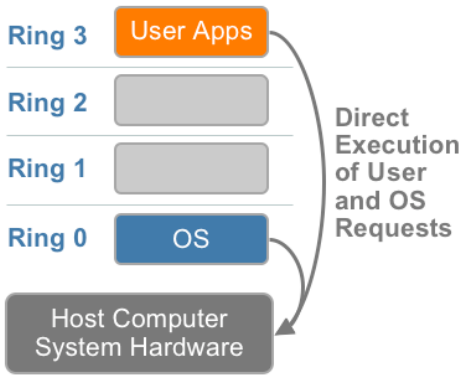

Figure 6.1: Non virtualized system[45]

Figure 6.1 show the path of instructions for a non virtualized system. x86 operating systems assume they are running directly on top of the computer hardware. The x86 architecture provides four levels of privileges called *rings*. As seen in the figure, the operating system is executed at ring 0, the ring which allow calling hardware and memory instructions (privileged instructions). These instructions have different semantics when not executed from ring 0. The problem with virtualization is to execute privileged instructions while not in ring 0.

Software Virtualization

With software virtualization, each virtual machine instance is provided with virtual BIOS, devices and memory management by the virtual machine monitor. Privileged instruction calls get caught and translated into sequences of instructions which together execute with the same intended behaviour. This technique, as seen in figure 6.2, is called binary translation. User application instructions are executed directly on the hardware and does not need to be translated. This technique provide full virtualization as the guest OS is fully decoupled from the underlying hardware.

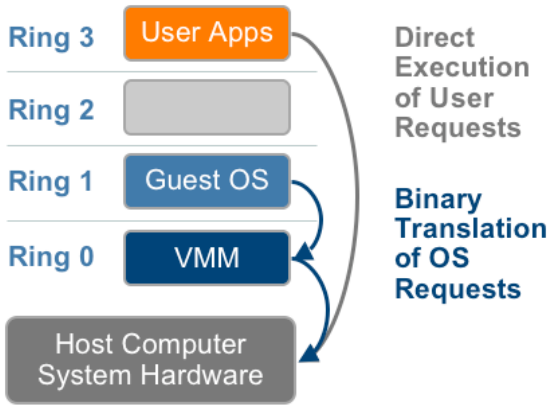

Figure 6.2: Binary translations on privileged instructions[45]

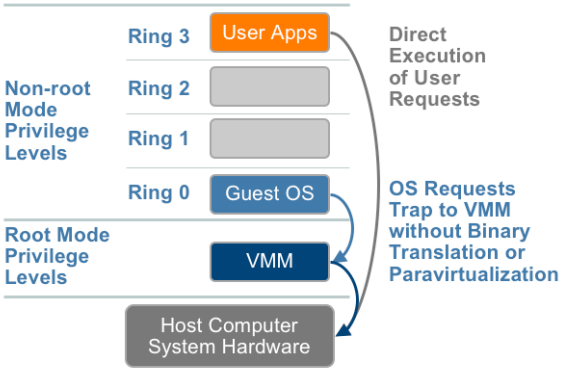

Figure 6.3: Hardware virtualization[45]

Hardware Virtualization

To improve the virtualization performance, several additions had to be added to processors and chipsets. In 2006, AMD and Intel added support for hardware virtualization (figure 6.3) with their technologies AMD-V and Intel-VT-x. The current hardware virtualization is divided into three components.

The first and most important component is CPU virtualization; trapping the privileged instructions from the guest OS and emulate them on the CPU.

The second component is memory virtualization, sharing the physical memory with virtual machines.

The thrid and final component is Device and I/O virtualization; routing I/O instruction requests between virtual devices and physical devices.

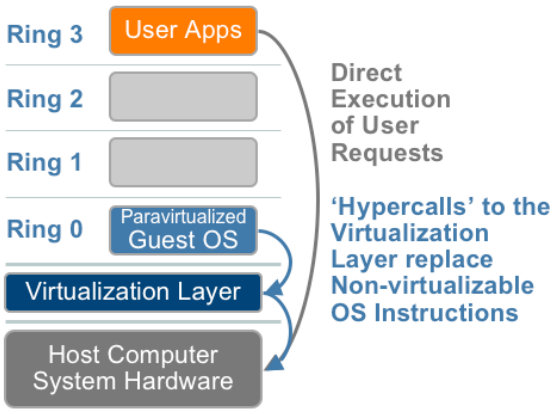

Figure 6.4: Paravirtualization[45]

Paravirtualization

Paravirtualization is a technique where the guest OS uses hardware drivers specialized for the underlying VMM. The guest OS must also be modified to run on the VMM. Non-virtualizable instructions are replaced by *hypercalls* in the OS kernel. Hypercalls are communicating directly with the hypervisor providing the virtualization layer.

6.3 Virtualization in IaaS

Virtualization and virtual machines are key components of Infrastructure as a Service in terms of providing virtual hardware. As displayed by figure 5.1 Cloud OSs balance the load of virtual machines on top of physical hardware and provide networking and other resources for the VM instances.

Cloud OSs also provide possibilities of pausing and suspending instances, migrating instances from one physical machine to another, or even to different data centers, duplicating instances etc. All in all Cloud OSs provide ways to manage and operate virtual machine instances on multiple physical machines.

6.4 Virtual Appliances

A virtual appliance is a virtual machine image designed to run on a virtualization platform, designed to eliminate the maintenance cost of running complex software stacks, and to minimize the need for configuration.

Virtual appliances are distributed with Just Enough Operating System(JeOS)[46], a customized and minimal OS that fits the needs of a particular application. They are normally distributed as a pre-configured disk partition with no need for OS

installation. This approach give advantages such as a homogenous environment for the patricular application, simplified VM deployment and improved isolation for applications.

Virtual appliances is just what is needed for our solution. The homogenous isolated environment can prevent AliEn from any unnecessary interaction and relations to other services, and at the same time give only one platform to support. Its minimal nature can potentially give a short startup process. The remaining dependency is tailoring the appliance for a specific software application, in this case AliEn. CernVM together with CernVM-FS (CernVM File System), as we will see in the following sections, will provide what's needed to fulfill the last dependency.

6.4.1 CernVM

CernVM[7] is a virtual software appliance designed to provide a runtime environment for the LHC experiments. It is built using a commercial tool for automating application deployment on virtualized or physical resources by rPath, called rBuilder. This tool gives software providers the capability to quickly pack applications with a minimal OS. The minimal OS use the Conary Package Manager.

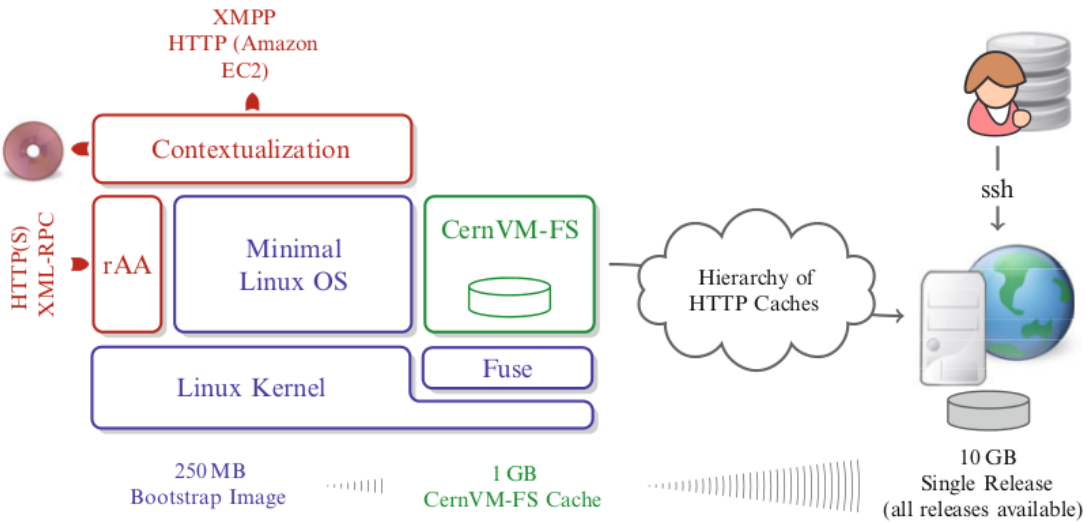

Figure 6.5: CernVM consist of, among others, a minimal OS (JeOS), components for contextualisation and the CernVM File system[47].

The idea of CernVM is to reduce the number of platforms to support and minimize time used for setting up experiment environments. In figure 6.5 we see the CernVM components. Its minimal OS is a modified version of Scientific Linux, a Linux distribution based on Red Hat Linux. The Contextualization component provides a way to specify the purpose of a CernVM instance through different parameters. These

parameters can be used to e.g. start services, create users and other configuration parameters. A script may also be executed on startup using the contextualization component.

6.4.2 CernVM File System

As a way of generalizing the OS while still being highly customized, CernVM can be configured to use different software repositories with pre-built software.

CernVM-FS (CVMFS) is a HTTP-based read-only file system. When accessing a folder, only stubs for the files are downloaded so that they can be listed. The actual file contents will only be downloaded when accessed directly. From a user perspective it behaves like a regular read-only UNIX file system. Files can be accessed by "/cvmfs/server/path/to/executable" where *server* is the DNS name of the CernVM-FS host.

Another benefit is that CVMFS is currently being installed on all AliEn sites, replacing torrent as the install method for software packages[27]. The more widespread the system is, the more supported it will be considering quality of servers, bandwidth and number of available packages.

In addition to the benefits described above, this technique allows the OS to be minimal. Most of the software not used in the startup process can be removed from the distributed virtual appliance, and instead be downloaded when needed.

For this project, the contextualisation parameters can be used to tell CernVM to add alice.cern.ch as a cvmfs repository, giving CernVM access to all the AliEn software appearing as it is already installed, when in fact it is being downloaded when needed.

6.4.3 Why use CernVM

The Contextualization feature is of particular interest for this project as it provides a way to specify which software sources to use for an instance without editing the image itself. By giving the instance a list of parameters, the instance will be customized on boot up. In addition to these parameters a bash script, which will be executed on start up, can be added to these contextualization data. A more detailed description of the importance of this feature will be given in chapter 8

ALICE is hosting a CernVM-FS software repository in which AliEn can be found at "/cvmfs/alice.cern.ch/". By accessing AliEn this way, the contextualisation script can be simplified by treating the system like AliEn is already installed, where the alternative would to use other installation methods like *wget* to grab alien-installer

which use torrent installation etc.

From a Grid perspective CernVM provide a homogenized execution environment, easing distribution of software packages. At the same time, as a natural advantage of virtualization, CernVM provide isolation from the underlying host system, giving confidentiality of data.

6.4.4 µCernVM

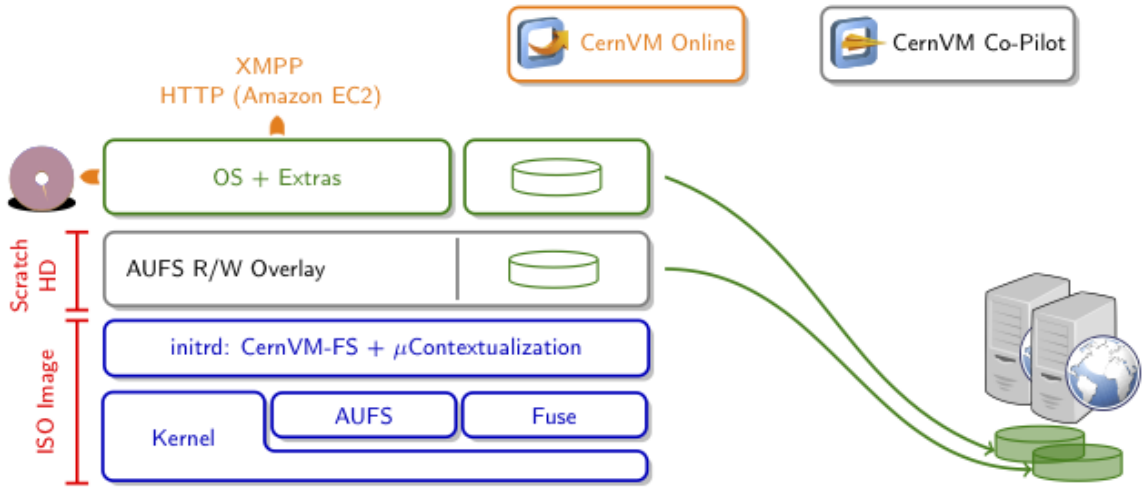

Figure 6.6: Overview of the components of μCernVM[48].

µCernVM (micro CernVM) is the next generation CernVM or CernVM 3.0. It is distributed as a 10MB disk image containing only the linux kernel and the CernVM-FS client as seen in figure 6.6. The rest of the OS is downloaded and cached on demand by CernVM-FS. Downloading all required software can take time depending on the internet connection speed. To solve this each cluster can have a local cache so that the software is only downloaded from an external source once. Downloading from a local source is normally faster than from an external. Although some software must be downloaded, only the software actually used will be downloaded instead of having unused software, and with a local cache the download time will generally be short.

Although μCernVM will not be used for our project, it is an interesting piece of technology which should be tested in combination with the proposed solution in chapter 8 (will be discussed in section 10.3).

6.5 Copy on write

Virtual appliances are disk images prepared to be run on a hypervisor. In the case of CernVM this image contain a disk partition which is expanded to 9.7GB when it is mounted. Creating multiple VM instances would normally include the work of copying the whole partition for each instance, and with almost 10GB to copy it requires a lot of I/O operations which usually is a slow process and thus would significantly increase VM startup times.

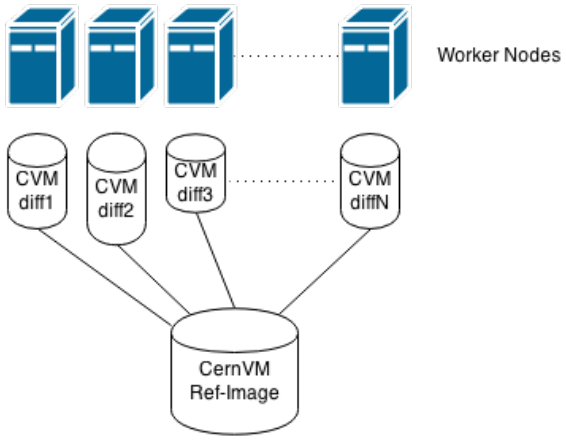

Figure 6.7: Copy on Write reads unmodified data from a source image and stores modifications on a differencing image.

Copy On Write (COW)[49] is a solution to this problem. This technique uses the disk image as a reference image and all changes done by the VM instance are written to a differencing image. The differencing image contains all changes from the reference image, starting at close to zero bytes and increasing based on the changes done by the VM instance. As seen in figure 6.7 this leaves just one of the original image (reference) and one differencing image per VM instance, saving a lot of I/O, storage space and time.

Chapter 7

System setup

Earlier chapters have described the concepts and technologies which is the foundation for this thesis. The final implementation of the proposed solution requires that these technologies are installed and correctly configured. This chapter will describe the setup of the system on which the prototype will be implemented.

7.1 Testing environment

A test environment for the designed prototype has been set up. This section will describe the process of installing the software systems described in the previous chapters with some discussion about choices of extra software and configurations.

7.1.1 Hardware

The test environment consists of 4 similar computers, as displayed in figure 7.1. One computer runs the central services and storage element of AliEn. Another computer is running as VO-Box, Compute Element, Cloud host, and Cloud compute node. The last two computers are running as extra OpenStack compute nodes, adding up to a total of three compute nodes.

CPU	Intel i5 2500k @ 3300 Mhz
CPU cores	4
RAM	4096MB DDR3
RAM Clock	1600MHz

Table 7.1: The testing environment consist of four machines with similar hardware setup.

This setup gives the possibility of hosting 12 VM instances with 1GB RAM and

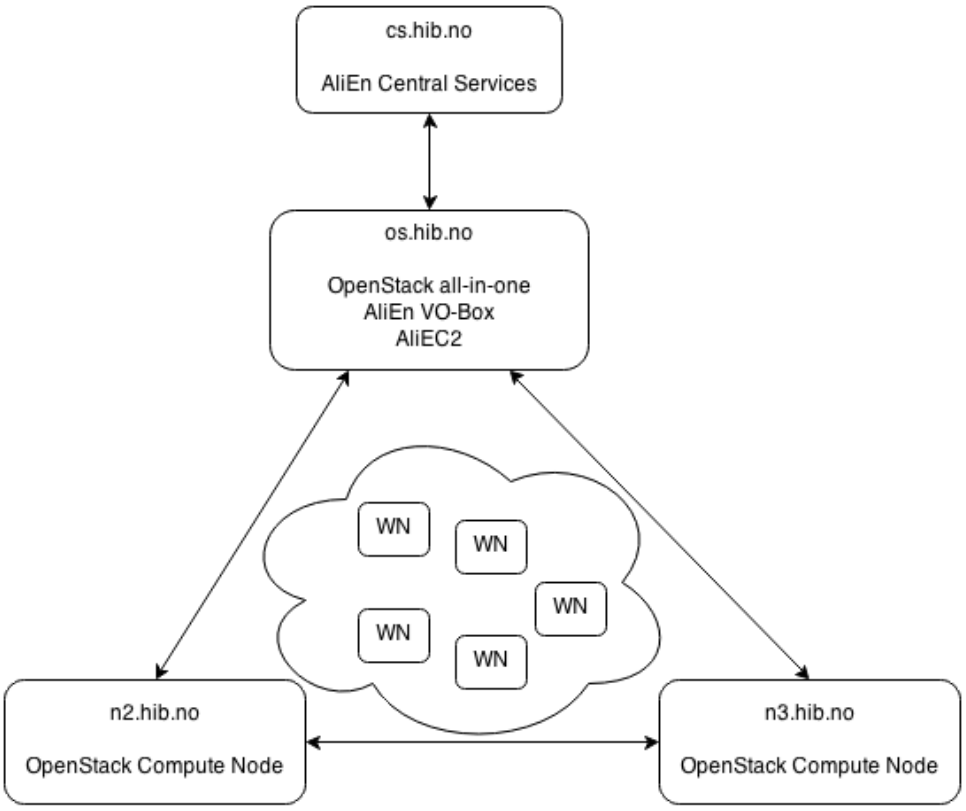

Figure 7.1: The different machines within the testing environment. They are named os, cs, n2 and n3 and are in the domain *hib.no*

1 VCPU on the three compute nodes. This amount of virtual resources should be sufficient to test the VM instance lifecycle management of the proposed solution (will be introduced in section 8.2).

7.1.2 Software

The software versions used in this project will be described in this section.

- **Host OS:** The physical machines in the testing environment run Scientific Linux 6 (SL6), a derivative of Red Hat Enterprise Linux 6 (RHEL6). Although some resources in the AliEn Grid are still using SL5, over the last few years most AliEn sites have transitioned from using SL5 to SL6, making SL6 the most natural choice of version to use[27].
- **Grid middleware:** AliEn v2-19.233 released on December 16th 2010 is the most recent stable release of the AliEn Grid middleware.
- **Virtual Appliance:** CernVM v2.7.1 released on October 14th 2013, late in the projects course. Initially CernVM 2.6 was intended to be used, but

an error prevented the contextualisation data to be loaded correctly. The component responsible for contextualisation, amiconfig, was unable to detect the OpenStack EC2 user data URL. Without this URL, the EC2 user data could not be downloaded and executed, impairing an important part of the designed solution for this project. After some investigation, the error turned out to be a variable in the amiconfig scripts which never got assigned when the VM was hosted on EC2 clouds.

- **Cloud OS:** OpenStack Grizzly was used while developing the solution, although two new stable releases called Havana and Icehouse have been released in the meantime.
- **Hypervisor:** As hypervisor used with OpenStack, the Kernel Virtual Machine (KVM) was selected because KVM is the hypervisor which RHEL6, and thus SL6, have chosen to support.

EC2 and OpenStack uses different conventions for giving IDs to instances, images etc. OpenStack uses random hex-strings, while EC2 uses a component identifier followed by a unique number e.g. ami-00000001, where *ami* means *image* and the following number represent the unique incremented image ID. The EC2 ID format is needed when using the EC2 API. When using the OpenStack Command Line Interface (CLI), the EC2 ID is not shown. Therefore the eucalyptus CLI *euca2ools*[50], a CLI for Amazon Web Services(AWS) compatible web services, will be used to get the EC2 IDs.

7.1.3 Libraries and frameworks

This section will describe the range of libraries and frameworks used for implementing the prototype.

- This project have used an open source EC2 library created by Lincoln Stein, at Ontario Institute for Cancer Research, called LibVM-EC2 (VM::EC2)[51]. LibVM-EC2 did initially only support EC2 endpoints using the default EC2 API port. A small change to the code was requested[52], enabling support for OpenStacks default EC2 endpoint. This change allows for altering the port used to connect to the EC2 endpoint.
- For loading configuration files the Perl library Config::Simple have been used. This library provides a simple way of reading and storing configuration parameters.

- Perl Dancer[53] have been chosen for developing the AliEC2 (will be introduced in section 8.5.1) web service. It was chosen for its simplicity and its good reputation.
- For interaction with databases using Perl, Perl Database Interface (DBI) was selected, as it is the standard database interface module for Perl. It has multiple extensions for different databases like MySQL, SQLite, PostgreSQL etc.
- The database software used for the web service is SQLite. SQLite is a serverless SQL database engine which require no configuration, and is thus convenient for development purposes. The implementation can use any SQL engine, so SQLite is not required and is easily replaceable in the configuration (see section 8.5.3.)
- For contacting the web service from AliEn::LQ::EC2 (will be introduced in section 8.5.2) Net::Curl::Easy was used along with Net::Curl::Form. Note that the latter require the libcurl-devel package to be installed.

All the Perl libraries and frameworks are made available by CPAN (Comprehensive Perl Archive Network)[54] as VM::EC2, Config::Simple, Dancer and DBI (with extensions like DBD::SQLite and DBD::MySQL) respectively (see appendix A). Other dependencies which must be installed (also available through CPAN) are DateTime, DateTime::Format::DBI, Log::Log4perl, DateTime::Format::SQLite, Dancer::Plugin::Form and Dancer::Logger::Log4perl.

7.1.4 Virtual machines

CernVM is distributed in different formats based on hypervisor and on intended use. As KVM is selected as hypervisor on OpenStack and the CernVM instances are supposed to run as batch nodes, the CernVM batch node release for KVM was the natural choice. CernVM instances are created using the qcow2 format (a Copy on Write implementation as described in section 6.5).

7.2 Installation

This section will describe the process of setting up OpenStack, AliEn, CernVM and CernVM-FS, and describe the most important configuration parameters used.

7.2.1 OpenStack

OpenStack has a lot of different components which could be installed on different computing resources. As the installation for this project is not on a production site,

this setup can be simplified by installing all components on one resource, called an all-in-one setup. Two extra compute elements will also be set up to allow more virtual machines.

All-in-one

The widespread use of OpenStack has lead to a range of automated setup tools, making the OpenStack installation process painless. However none have greatly simplified the installation on multiple nodes.

PackStack is a tool simplifying the deployment of OpenStack. It provides a simple way of installing an all-in-one setup which is exactly what is needed for this project. As mentioned in section 7.1.1, the all-in-one node will be running on the same machine as the VO-Box. The installation of the last two machines running as OpenStack compute node only, will be covered in the next subsection.

The *packstack* command with the option *--generate-answer-file* creates a file with all parameters needed to setup all openstack components, everything from default user of the system to the login on the MySQL database used by OpenStack. The passwords in this generated file are random strings, so all these can be replaced making them easier to remember. Some of the most important configuration parameters are listed in table 7.2.

CONFIG_MYSQL_USER	Username of the mysql admin user. Important if mysql is already installed.
CONFIG_MYSQL_PW	Password of the mysql admin user. Important if mysql is already installed.
CONFIG_KEYSTONE_ADMIN_PW	The password of the admin user of OpenStack
CONFIG_NOVA_NETWORK_FLOATRANGE	The IP range for virtual machines.

Table 7.2: PackStack have a wide array of configuration parameters. These are the most important variables to change manually for an all-in-one setup.

The next step is to actually install the components. PackStack provides the command *packstack --all-in-one --answer-file=theAnswerFileCreatedAbove*. This process usually takes some time, as there is a lot to install.

When the installation step is done, a new file called *keystonerc_admin* will appear in the root user's home folder. This is a bash-file exporting environment variables used for authenticating when working with the OpenStack client tools. When working with the command line interface of OpenStack (some of the CLI tools

used are called nova, glance, cinder and keystone) this file must be sourced (*source keystonerc_admin*) to provide the environment variables needed to operate the CLI.

Additional Compute Nodes

The hardware described in subsection 7.1.1 uses 3 OpenStack compute nodes. During the all-in-one installation, only one compute node is set up. Adding compute nodes to OpenStack can be done in a few simple steps by editing some variables in the packstack answer file. First the name of the private network interfaces for the host (the all-in-one node) must be changed (CONFIG_NOVA_NETWORK_PRIVIF and CONFIG_NOVA_COMPUTE_PRIVIF) from loopback (lo) to the name of a network card (i.e. eth0). Additionally, the IP addresses of the new compute nodes must be added to the comma separated list CONFIG_NOVA_COMPUTE_HOSTS, and it must be ensured that the IP address of the host is contained in the key CONFIG_NOVA_NETWORK_HOSTS. After applying these changes, the packstack tool with the edited answer file needs to be re-run. The new compute nodes are installed and added to the cloud system.

EC2 Credentials

By default, OpenStack uses a username and password authentication model. To be more flexible when it comes to supporting different Cloud Operating Systems, EC2 was chosen as the API to use. EC2 is having its own authentication model with access keys and secure keys, both 32 character hexadecimal keys. To be able to use the EC2 API programatically with LibVM-EC2, these keys must be created. By providing the tenant-ID and user-ID of the OpenStack user, the command *keystone ec2-credentials-create --user-id=$USER-ID --tenant-id=$TENANT-ID* will create EC2 credentials for the user of the tenant. The values for user-ID and tenant-ID can be found respectively by *keystone user-list* and *keystone tenant-list*. For this project, the default user and tenant *admin* have been used.

The EC2 credentials generated are needed by the eucalyptus CLI tools and should be added to the keystonerc_admin file as EC2_ACCESS_KEY, EC2_SECRET_KEY and EC2_URL.

Virtual Appliance

CernVM is distributed in many formats. One of these formats are specialized for KVM. The image is formatted to the QCOW2 format, a copy on write format, after downloading. This image is uploaded to the glance image store where it can be

accessed and used when creating differencing images. The first virtual machine using this image will have a few minutes of extra startup time to prepare the image for copy on write, and because of reasons mentioned in the next chapter, this should be done manually outside the solution of this project.

Virtual CPUs	1
Memory	1024 MB
Swap space	2GB
Disk space	10GB

Table 7.3: The virtual resources given to the virtual machines for this setup

Although the CernVM batch distribution is small of size when distributed, when mounted on a virtual machine, it expands to about 9.7GB. The VM flavor selected for running CernVM should therefore have a disk size of about 10GB.

The physical compute nodes have 12 CPU cores and 12GB RAM when added up. Splitting these resources evenly on virtual machines will give the setup described in table 7.3, 12 VM instances with 1 VCPU and 1Gb RAM, or 9 if leaving one CPU and some RAM to each physical machines, a number which should be sufficient for testing the lifecycle management. Considering swap space, a general rule of thumb in the Linux community dictates that a Linux OS should have twice the amount of RAM as swap space, hence 2GB.

Security groups

Security groups were introduced in section 5.6.5, but are not necessarily required for this project. The worker nodes need no inbound ports opened, but for debugging purposes port 22 is opened to allow SSH connections. This only applies to this project, and will not be needed in a production environment. Anyhow, adding and removing security group rules is a trivial matter.

Keypairs

To allow SSH connections, an OpenStack keypair has to be created. An OpenStack keypair is specified by a keypair name and a public key. The public key is automatically added to the virtual machines authorized_keys-file by OpenStack, allowing passwordless SSH connections. As CernVM has no default password for the root user, it can not be accessed using a password without making any changes to the image itself, and therefore the OpenStack keypair can be used if SSH access to the

CVMFS_REPOSITORIES=grid.cern.ch,alice.cern.ch
CVMFS_HTTP_PROXY=DIRECT

Table 7.4: The file /etc/cvmfs/default.local must be created and this configuration must be added to it.

virtual machines is desirable.

7.2.2 CernVM-FS

Besides using CernVM-FS to install AliEn on the virtual worker nodes, CernVM-FS is used for running the AliEn services on the local site on the testbed. The CernVM website[48] provides the RPM packages required for installation, *cernvmfs-keys* and *cernvmfs-client*. The *cernvmfs-init-scripts* package provide a few helpful tools for debugging and setup verification. This package is not required for the installation, but is used in this project for its convenience.

After installing the packages, the client has to be configured to use the CVMFS repositories of ALICE (alice.cern.ch), grid tools (grid.cern.ch), and to set which proxy to use. The proxy configuration tell where to download data from.

Table 7.4 shows the configuration used on this project. The configuration file /etc/cvmfs/default.local must be created and the values listed must be added. Oddly, the mandatory variable CMVFS_HTTP_PROXY does not have *DIRECT* as default value. In fact it does not have a default value at all. *DIRECT* means that CVMFS download directly from the repositories specified instead of using another, possibly local, proxy server.

The init setup script, *cvmfs_config setup*, needs to be run after the CVMFS configuration have been done. When the init scripts are done, checking that the configuration is working correctly can be done by running *cvmfs_config chkconfig*.

7.2.3 CernVM contextualisation

As mentioned in section 6.4.1, the purpose of a CernVM instance can be specified by using a contextualisation file. The contextualisation file is composed of two parts; an executable script and a set of python style configuration blocks.

A part of the contextualisation file, which will be described in section 8.5.2, is the amiconfig contextualization parameters. This is a list of parameters and values grouped by plugin using the python configuration block style. There are a many plugins available within CernVM, but the most interesting is *rootsshkeys* which is enabled by default and the *cernvm* plugin. The plugin *rootsshkeys* allows injecting a

```
#!/bin/bash
# execute script here

./doSomething

#terminate the script by "exit"
exit

[amiconfig]
plugins = cernvm

[cernvm]
organisations = alice
repositories = atlas,alice,grid,atlas-condb,sft
shell = /bin/bash
```

Figure 7.2: An example of amiconfig used for contextualisation. It starts with executing an exit-terminated script followed by parsing python style configuration blocks.

public key into the *authorized_keys* file of the root user of the host OS. The plugin *cernvm* enable further contextualization parameters specific to CernVM. An example of amiconfig configuration can be seen in figure 7.2. The amiconfig block is used to activate or deactivate plugins.

The other part of the contextualization file is an executable script. The script will be generated as seen in the next chapter. The components used to generate the full script are available from github, see appendix A for the reference.

7.2.4 OpenStack and CernVM automated installation script

To simplify the installation process while trying different configurations of OpenStack together with CernVM, scripts for both automating the installation and uninstalling OpenStack with CernVM were developed. The script can be found and downloaded on github (see appendix A for the reference). It has two optional flags, -g for only generating the answer-file used by packstack, and -a for using an exsisting answer file. The script will ask for MySQL username and password, and requires sudo access.

The install.sh script installs and does basic configuration of OpenStack, creates a security group with a filter opening port 22 for instances. Moreover, it downloads the CernVM-image and uploads it to OpenStack, creates the CernVM volume, adds keypair for the user executing the script (it creates a pubkey if not found), and lastly it creates a CernVM instance and applies the settings from section 7.2.3.

The uninstall.sh script removes everything related to OpenStack, including software packages, openstack repositories, volume groups, configuration files and other

application data related to OpenStack. This script should be used carefully.

7.2.5 AliEn site

The introduction of CVMFS has eliminated the need for manually downloading and installing the AliEn site related software packages. Although the software is accessible, some configuration and setup must be done before it is useable. This process includes configuring the site services as well as the AliEn client.

AlienVO user

The front node of the site in this testbed is running the Compute Element (CE) and VO-Box software. These software packages are run as the user *alienvo* on the Linux system. To run the AliEn services as the *alienvo* user, some environment variables must be set.

ALIEN_ROOT	/cvmfs/alice.cern.ch/	Location of the root install directory of AliEn
ALIEN_HOME	/share/home/alienvo/.alien/	Location of AliEn configuration files and certificates
ALIEN_USER	aliprod	The site's user on AliEn.
ALIEN_NTP_HOST	hib-gsw.hib.no	The NTP server to use
ALIEN_DOMAIN	os.hib.no	The domain name of the machine.

Table 7.5: Some of the key environment variables for an AliEn VO-Box

The variables listed in table 7.5 show the configuration used in this project. The ALIEN_HOME directory contains the configuration for the AliEn site. The ALIEN_ROOT variable refers to the root folder of the AliEn installation, which in this project is in the CernVM File-System.

Startup configuration

AliEnUser	alienvo	The Linux user which run the AliEn services
AliEnCommand	"/cvmfs/alice.cern.ch/bin/alien"	Location of the AliEn command on the host machine, but also on the worker nodes.
AliEnServices	"Monitor CE CMreport"	The services to be run on this box.

Table 7.6: The file `$ALIEN_HOME/etc/aliend/ALIENBERGEN/startup.conf` contains the startup configuration for AliEn on the host machine.

AliEn startup configuration is specified in `$ALIEN_HOME/etc/aliend/startup.conf`. This file contains a variable called ALIEN_ORGANISATIONS which value is set to *ALIENBERGEN*. That value further correspond to a folder `$ALIEN_HOME/etc/aliend/ALIENBERGEN/` which contain another startup.conf file, which contain the startup configuration for AliEn on the host machine. One variable to note is the *AliEnCommand* variable which will be used when generating the job agent (JA) startup script to describe the location of AliEn on the worker nodes.

7.2.6 Central Services

The Central Services used in this project was already set up by one of the supervisors, Bjarte Kileng, leaving close to no work except a few operations for this project.

LDAP

INSTALLMETHOD	CVMFS	Which package installer the CE should use.
TYPE	EC2	Which batch system interface the CE should use.

Table 7.7: The key LDAP entries used for this testbed.

Lightweight Directory Access Protocol (LDAP) is a protocol for accessing static configuration parameters/directives remotely. This service is used to configure VOs, sites, and other services in the AliEn Grid. Settings like service URLs and ports, location of certificates, job management commands etc. are specified here. A large and complex LDAP database may be hard to maintain. Apache Directory Studio[55] provide a helpful way for doing so.

There are two key LDAP entries which values must be set as listed in table 7.7. The first LDAP entry important for this project is the *INSTALLMETHOD* entry of the CE of the testbed. The value of the entry must be set to "CVMFS", and is used for generating JA startup scripts on the CE as will be demonstrated in section 8.5.2.

The second important LDAP entry also for the CE of the testbed is the *TYPE* entry which must be set to "EC2". This value is used by the CE to choose which batch system interface to be used. As will be demonstrated in section 8.5.1, a new batch system interface is implemented, and the *TYPE* entry must correspond to the name of the new interface.

Environment Variables

Although most of the site configuration can be done using LDAP, there is also some configurations which can be done locally on the site. Most of these specify how the

system will interact with the users, like the shell used within AliEn, and paths to software installations. While testing different configurations, the CE can be locally configured by overriding the LDAP variables. This can be done by adding the prefix *CE_* to the name of the LDAP entries and add them to the alienbergen.conf file.

Chapter 8

Proposed solution

This chapter describes the requirements of a solution to the problem described in section 1.4, followed by an introduction to this proposed solution and how it fulfills the requirements. In the previous chapter it was shown how to set up and configure the systems needed to implement the proposed solution. Now remains the glue between these systems to complete the proposed solution.

8.1 Requirements

There are some requirements specified in the problem description in section 1.4 which the solution must satisfy.

- A virtual machine instance should exist only while executing a job. When a job is detected on the central services by a compute element on a site, a new VM instance should be started. Further on, when the job is done executing, the VM instance should be deleted. This approach will result in AliEn using computer resources only when needed, resulting in an elastic AliEn site.
- The proposed solution should have a minimal footprint on the existing AliEn Grid, both from a users perspective and the site administrators. A user should not be concerned about whether his/her job is running on a traditional AliEn worker node or on a virtual worker node in a Cloud system.
- A system administrator should have to do a minimum of changes to the existing system when implementing this proposed solution on a site. This requirement may vary from site to site depending on the systems already installed. Some sites may already have installed an EC2 enabled cloud system, and most sites already have AliEn set up.

- Minimal changes on the Central Services (CS), especially in the code running the Central Services, is required.

8.2 Idea behind the solution

This section will give insight into the idea behind the proposed solution. The proposed solution is split into two main components where the first is extending the AliEn CE, while the second is used to manage the lifecycle of the VM instances.

8.2.1 AliEn CE Type

AliEn sites have the option to choose which type of batch system to be used on a Compute Element. This can be specified in LDAP and be overridden by a local configuration on the VO-box. As explained in section 4.2.1 the CE is responsible for sending job agents to the local job queue on its site. To do so, the CE executes a function on a batch system interface, using a perl module which is imported into the code of the CE (AliEn::CE) based on the specified type of batch system. By creating a new module acting as a batch system interface, this function can be used to start virtual machine instances instead.

When the CE is started it looks for the specified batch system interface in the AliEn::LQ package and imports it. One of the purposes of the batch system interface is to handle the event when the CE has a job ready for scheduling. On this event, the *submit* function of the interface is executed. By overriding this function, the batch system interface can be used to request new VM instances.

Moreover, the AliEn::CE module contains a function for generating job agent startup scripts, a bash script for starting AliEn job agents on worker nodes. As mentioned in section 6.4.3 CernVM uses contextualization to adapt for different purposes. One way to achieve contextualization is to use EC2 user data (mentioned in 5.6.6). By adding the job agent startup script to the EC2 user data, the job agent can be started once the VM instance have started, contrary to actively accessing the VM instance through e.g. SSH to start a job agent.

8.2.2 Lifecycle management

As mentioned in the previous section, by creating a new batch system interface, AliEn can now request new VM instances as worker nodes. This will require some interface between the batch system interface and the EC2 enabled cloud system. This could be achieved either by using the EC2 API directly, or by creating another component

responsible for EC2 communication. Keeping in mind that the VM instances should be terminated when the job agent is done, the latter gives the possibility of creating a component responsible for both starting and terminating instances. The other alternative would have the batch system interface being responsible for starting instances and another component being responsible for terminating. By having one component responsible for EC2 communication, all EC2 functionality required for the proposed solution will reside in one component. Another alternative would be to stop the VM instances from the batch system interface, but that is impossible as the submit function of the batch system interface is dead after requesting a new worker node.

Termination of instances might seem like a trivial problem, but it is not. Events which might happen, like job software freezing, VM instances not starting, or other trouble with the virtual appliance which leave the VM instance in a zombie state, must be expected and handled. If not, VM instances may keep the resource indefinitely while in a zombie state.

To handle these problems a web service was decided to be the most fitting solution; a service running on the site with "clients" running on the virtual machines contacting the service by a specified interval. By doing this, the VM instances can tell the service that they are still alive and responsive. The instances can also tell the service to delete the VM instance when their Job Agent is done. Any virtual machine not giving an update within the specified interval can be assumed dead and be deleted. The service can also be contacted for aquiring new VM instances, essentially managing the lifecycle of the instances related to AliEn. The service will from now on be referred to as AliEC2.

8.3 Summary

Adding the requirements from section 8.1 with the idea from section 8.2, we get a list of guidelines for the implementation.

- The worker node should tell HTTP service it is still alive. Service will kill the VM if no response.
- Worker node should tell the service when it is done.
- The service should delete non responsive machines after specified period. This period should be sufficient for a VM to start up and should be configurable.
- The solution should be transparent for end users. Users of the AliEn Grid should have no need for being aware of the changes done to a site.

- The service can be contacted for requesting new worker nodes.

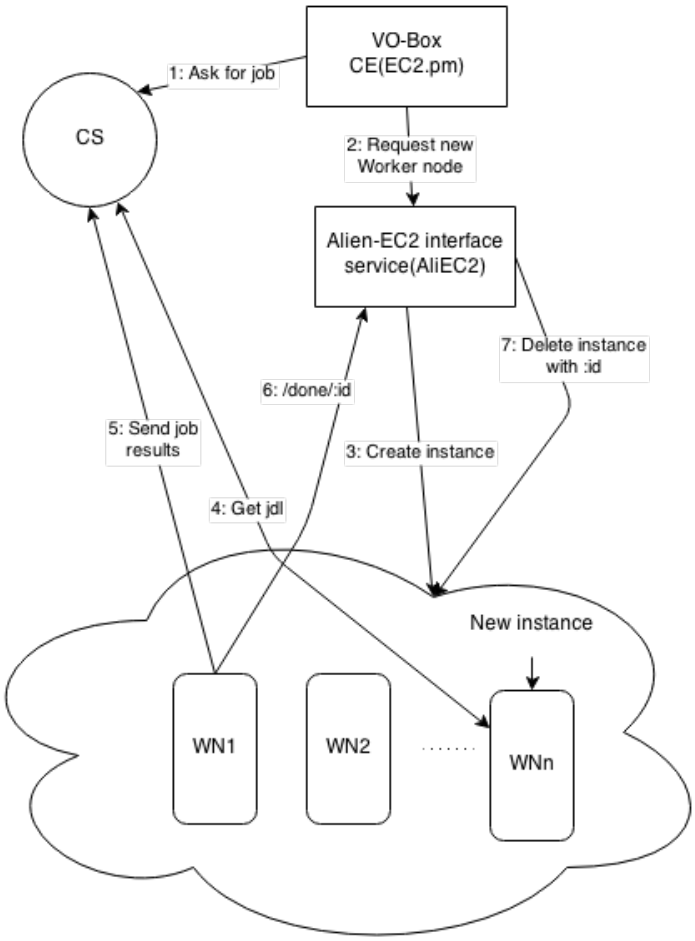

Figure 8.1: The web service act as an interface between the AliEn batch system interface and the underlying EC2 enabled cloud system (OpenStack in this case). It is responsible for managing the lifecycle of the VM instances.

The implementation will work as described in figure 8.1. The batch system interface module discussed above will contact the service when new resources are needed. The service will through the EC2 API request a new CernVM instance from OpenStack and transfer the Job Agent startup script to the new VM instance by using EC2 user data. The AliEC2 service will at specified intervals look for and kill zombie instances, instances which have not told the service that it is alive. Instances contacting the service because its job agent is done will also be deleted immediately. Moreover, as displayed in figure 8.1 the two components have been named AliEn::LQ::EC2 as the batch system interface, and AliEC2 as the AliEn-EC2 interface service.

8.4 Design decision

There are multiple ways of implementing the proposed solution, some more fitting than others. This section will describe why the design described in the above sections was chosen, and compare it with other possibilities.

Early on, the plan was to locate events in the AliEn system, one event telling that a job have been submitted, and the other event telling that a job has been finished. The first event have been located, but the second event could not be found and it was later on confirmed by the AliEn team that no such event existed. With no such event, a way to monitor the job agents and the VM instances was needed. A way to implement this is to have a service on the site with active clients contacting the service and updating their status.

Another option would be to have passive clients with a service making queries. This approach would require more open ports on the clients, and also downloading of libraries and frameworks for hosting an endpoint which the service could contact asking for the job agent status.

For this purpose the first option, a web service with active clients updating their status, seemed the most obvious solution with the least drawbacks. Moreover, as mentioned in section 8.2.2 the service may also be used for creating new VM instances, and thus it gathers all EC2 communication in one component.

8.5 Implementation

The layout for the system displayed by figure 8.1 has been implemented and set up on the testbed described in chapter 7. This section will give a more detailed description of the implementation of the two components described in section 8.2.

Most of the programming for the implementation has been done using the Perl programming language, but also some bash scripting was used for the contextualisation scripts of the virtual machines. In addition, some SQL was used in the AliEC2 Web Service component for handling the database.

8.5.1 Lifecycle Management Service (AliEC2)

The lifecycle management service, named AliEC2, has two main functions. The first main function is acting as a web service, and the second is running an update loop for checking the state of the VM instances and delete zombie instances. A thread for each of these functions are created when AliEC2 is started.

GET /alive/:id	VM with ":id" tell the WS it's alive.
GET /done/:id	VM with ":id" tell the WS it's done and can be deleted.
POST /spawn	AliEn::LQ::EC2 uses this to request a new VM instance. POST parameter "script" can be used to provide EC2 user data.

Table 8.1: The web service have a set of functions which can be contacted by the VM instances and the new AliEn::LQ module.

The first main function, the web service component, was implemented as a web service (WS) using the Perl Dancer web framework. Three HTTP functions are needed as listed in table 8.1. Figure 8.2 shows the flow of the different web service functions.

The first function (/alive/:id) is used by the VM instance to tell the WS that it is still alive and responsive. The second function (/done/:id) is also used by the VM instance to tell the WS that its job agent is done and that the instance can be terminated. The last function (/spawn) is for the AliEn::LQ::EC2 (see section 8.5.2) batch system interface to request a new VM instance. The agent startup script generated by the CE code (as mentioned in section 8.2.1) should be supplied as POST data.

The communication with EC2 is extracted to its own module AliEC2::EC2 (not to be confused with AliEn::LQ::EC2) to group the EC2 functionality in its own module. The web service uses functions of the AliEC2::EC2 module to create and delete VM instances. AliEC2::EC2 uses VM::EC2 (LibVM-EC2, as mentioned in section 7.1.3) for EC2 communication.

The second main function is running a loop within a specified configurable interval (e.g. every 60 second), iterating over the instances listed in the database looking for instances which have not given an update within that interval. The database will be described in more detail later in this section. The update loop differentiate between recently spawned instances and instances which have already given an update. Instances recently spawned are given a larger interval to ensure that it is not deleted while starting up.

As mentioned in the beginning of this section, the web services component use the Dancer framework for handling HTTP calls. Each HTTP call forward their job and instance ID parameters to a database handler module (AliEC2::DB). AliEC2::DB use the Perl Database Interface (DBI) which supports multiple database types. While this project uses SQLite (DBD::SQLite), in theory any database supported by DBI could be used.

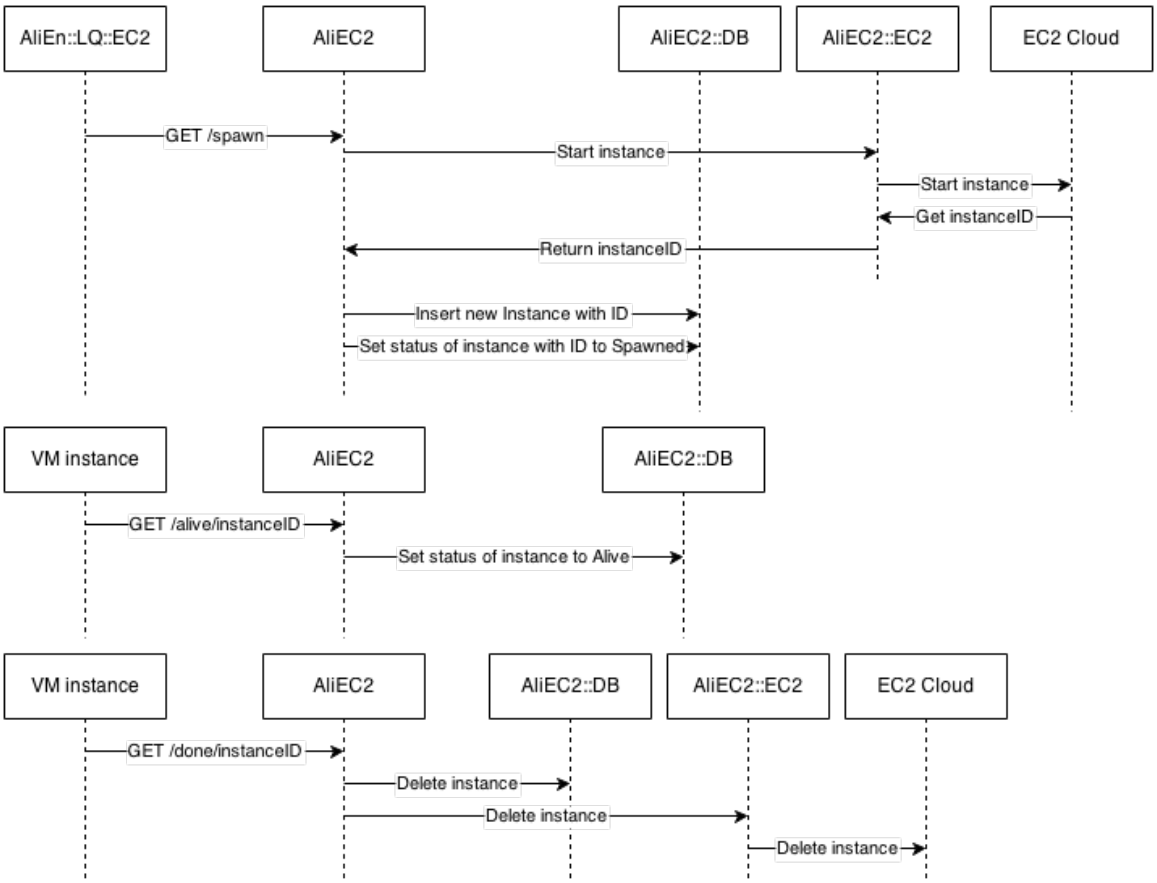

Figure 8.2: A sequence diagram displaying the actions and components behind each function of the web service.

The database (Table 8.2) is used to store the state and time of the last update of each VM instance. The state can be either 'Spawned' or 'Alive' to differentiate between newly spawned instances and instances which have updated their status at least once. Newly spawned instances may have a longer time limit to send a heartbeat before they are deleted.

Summed up, the AliEC2 service is composed of the following parts:

- **AliEC2**: The main module of the AliEC2 service. It initiates and starts the web service functionality of the AliEC2 service, as well as running the update loop.

- **AliEC2::EC2**: This module is responsible for all EC2 communication. The AliEC2 module uses this module for EC2 communication.

- **AliEC2::DB**: This module is responsible for handling the database.

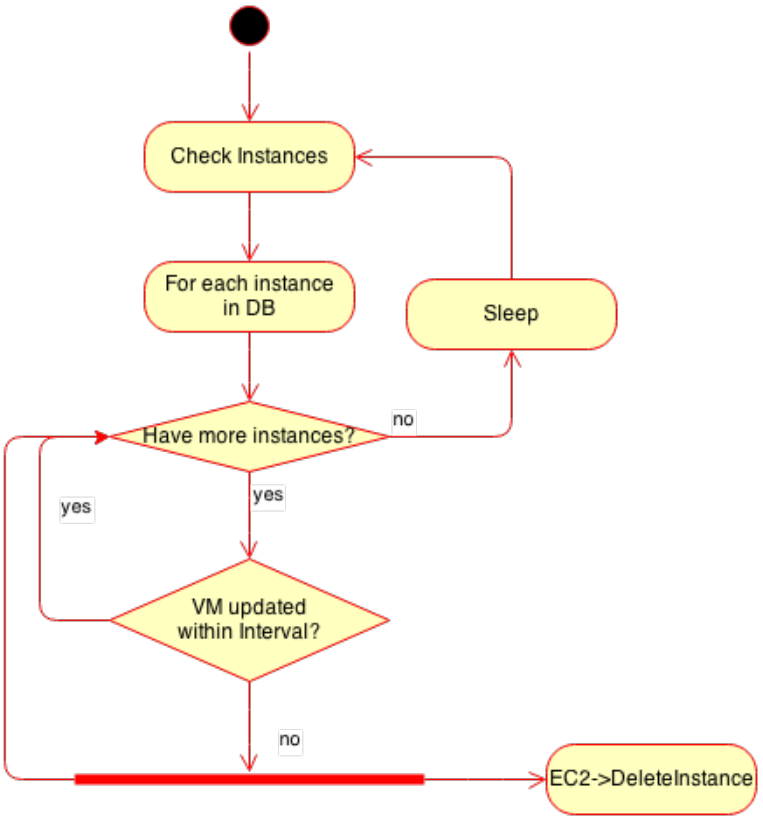

Figure 8.3: The update loop is looking for zombie VM instances. A zombie is an instance which have not updated within the update time limit.

8.5.2 AliEn::LQ::EC2 batch system interface

When the AliEn CE detects a job on the CS, it calls the *submit* function on the specified AliEn::LQ batch system interface module. By creating a new AliEn::LQ module, the submit function can be overridden and used to start new VM instances. This new module is called AliEn::LQ::EC2, not to be confused with AliEC2::EC2 of the lifecycle management service.

The implementation of AliEn::LQ::EC2 is a modified version of AliEn::LQ::PBS (the batch system interface for PBS/TORQUE) where the submit function has

Field	Type	Description
InstanceName	Text	ID of an instance.
LastUpdate	Timestamp	Time of last instance update.
Status	Text	Status of the VM. Can be "Spawned" or "Alive"

Table 8.2: The different fields of the AliEC2 database.

been replaced. The submit function starts by generating the EC2 user data used to contextualize the CernVM instances. The process of generating the EC2 user data will be described later. Because the job agent startup script depends on the AliEn package installer of the CE and on the *AliEnCommand* of the AliEn startup configuration (as mentioned in section 7.2.5), the job agent startup script, and thus the EC2 user data may be different from site to site and must be generated when requesting a new virtual worker node.

The EC2 user data is generated by concatinating two files containing the parts of the contextualization data which dont need to be generated per worker node. In between the concatination, the job agent startup script is inserted. The JA startup script is generated by an existing AliEn function based on the selected package installer of AliEn, in this case CVMFS. After generating the EC2 user data, AliEn::LQ::EC2 contacts the AliEC2 web service by the /spawn HTTP function, requesting a new VM instance. The EC2 user data is attached as POST data within the HTTP request.

The contextualisation script is executed when the VM instance is started. It starts with exporting some necessary AliEn environment variables. Further on it starts the JA startup script in a new process so that it will not block the rest of the VM startup process. When the job agent is done, the script then contacts the AliEC2 web service through the /done/:id function asking to get the VM instance killed.

In parallel on startup, the contextualization script creates a new thread for monitoring the job agent. This thread regularly checks that the job agent process ID exists. If it exist, the thread contacts the Web Service on the VO-Box to show that it is still working and is alive. If not, the thread assumes the job agent is dead and tells the WS that the VM instance can be killed.

8.5.3 Configuration

The setup requires some configurations specific to the AliEn installation and the OpenStack setup used in this project. The two new components, AliEC2 and AliEn::LQ::EC2, must be configurable both for testing different configurations and for adapting to different environments. The EC2 API requires a specific URL and needs both an access key and a secret key for authentication. In addition to these, parameters for time limits and update intervals can be specified. Table 8.3 shows an example configuration.

AliEC2 stores the status in an SQL database. The example in table 8.3 uses

ec2_endpoint	http://localhost:8337/2007-15-12/	EC2 endpoint URL
ec2_access_key	abcdef1234	EC2 Access key
ec2_secret_key	4321fedcba	EC2 Secret key
check_interval	120	The interval of which the service look for zombie instances.
spawn_time_limit	300	The maximum time an instance is given to start up.
alive_time_limit	120	The interval a started instance must reply within before it is considered a zombie instance.
db_type	SQLite	Type of database. Corresponding to the libraries DateTime::Format (::SQLite) and DBD:: (SQLite) mentioned in section 7.1.3.
db_host	*blank*	Database host. As SQLite is used, it is local only.
db_name	aliec2.db	Database name. A filename in the case of SQLite.
db_user	*blank*	Database username.
db_pass	*blank*	Database password.
aliec2_host	10.0.0.35:8000	AliEC2 WS host IP address.

Table 8.3: Configuration parameters used by AliEC2 along with a description of each parameter and an example value.

an SQLite database file with no authentication for simplicity. This can easily be changed to use an already existing database. AliEC2 will create the tables needed.

In case of a CVMFS installation of AliEn on a site, the new AliEn::LQ module can not be added to the AliEn/LQ folder where AliEn normally looks for CE implementations, because CVMFS is read-only. There are two solutions to this problem. The first one is to put the new module in its own folder and add the folder to the PERL5LIB environment variable. The second one is to use *mount --bind* to override the folder pointer in the file system. This way the folder containing AliEn::LQ::EC2 can override the location where AliEn look for AliEn::LQ modules.

8.6 Performance

In this section the extra overhead caused by the proposed solution will be studied. The software implemented in this project will have a minimal impact on the overhead. Instead the systems used, OpenStack and CernVM with KVM along with CVMFS, will have the biggest impact on the performance. Additionally, components which could be further improved are identified in this section.

The lifecycle of the VM instances can be split into four parts: building the instance, boot up, execution of jobs and termination. Measuring the performance of each part individually allows for easier identification of weaknesses in the system. If building the instance is especially time consuming, the cloud system or the hypervisor is the probable cause. If the boot process is taking a long time, the hypervisor or the virtual appliance is the probable cause.

8.6.1 Building instances

After requesting a new VM instance, the instance is in the 'BUILD' state. Building instances is a matter of creating a new virtual machine, creating a differencing image from the source image and other steps such as assigning IP-addresses, setting up port filters; steps done by the hypervisor and the cloud OS. Measuring the time consumed when building VM instances is a matter of recording the time when the VM instance was requested until the state of the instance change from 'Build' to 'Active'.

To measure the build time of a VM instance, the timestamp just before the instance was requested and the timestamp just after the instance get the 'Active' state, was recorded. Figure 8.4 show the build time of instances on the testbed. The average build time is 29.8 seconds while the maximum is 33.8 seconds and

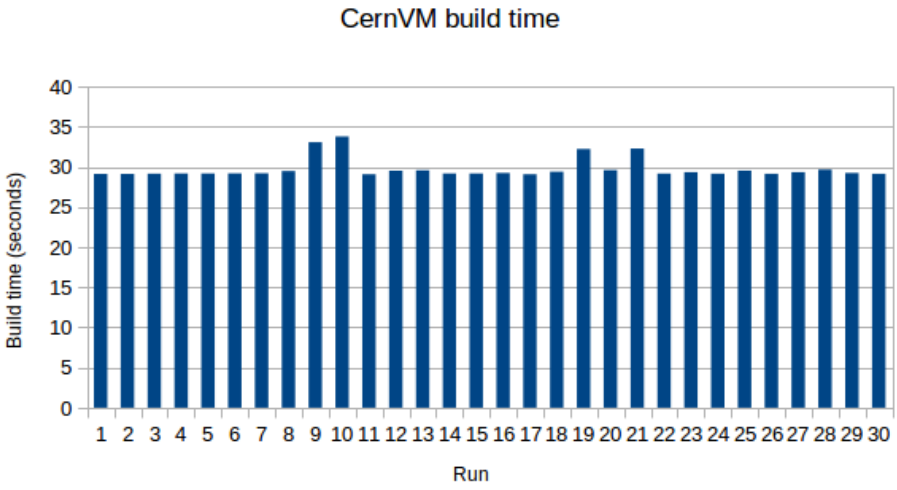

Figure 8.4: Building instances on the testbed on average take 29.8 seconds with a minimum of 29.1 seconds and a maximum of 33.8 seconds

the minimum is 29.1 seconds. The variation from minimum to maximum is of 4.7 seconds.

8.6.2 Startup

After the status of the VM instance have transitioned from 'Build' to 'Active', the boot process of the Guest OS is starting. To find the time consumption of this process, the interval between the instance's status was set to 'Active' and until the instance could access the /alive/:id function on the AliEC2 service was measured. To access the /alive/:id function, the contextualization script was used. The execution start of the contextualization script marks the end of the startup process and is when the job agent is starting.

Figure 8.5 shows that starting instances takes on average 92.6 seconds. The maximum and minimum boot time took 96.7 seconds and 88.8 seconds respectively, with a maximum difference of 7.9 seconds. In total the startup and build have a worst case of 130.6 seconds and best case of 117.9 seconds with an average of 122.3 seconds.

As the startup step was the most time consuming, some investigation was done in this step to understand the time consumption. A particularly interesting contribution to the time consumed is a file system check happening for each instance, a process measured to consume about 5 seconds. As the instances for this project are disposable, no changes have been done to the filesystems before the instance has started and should not require any checking. This step should not have to be executed and could

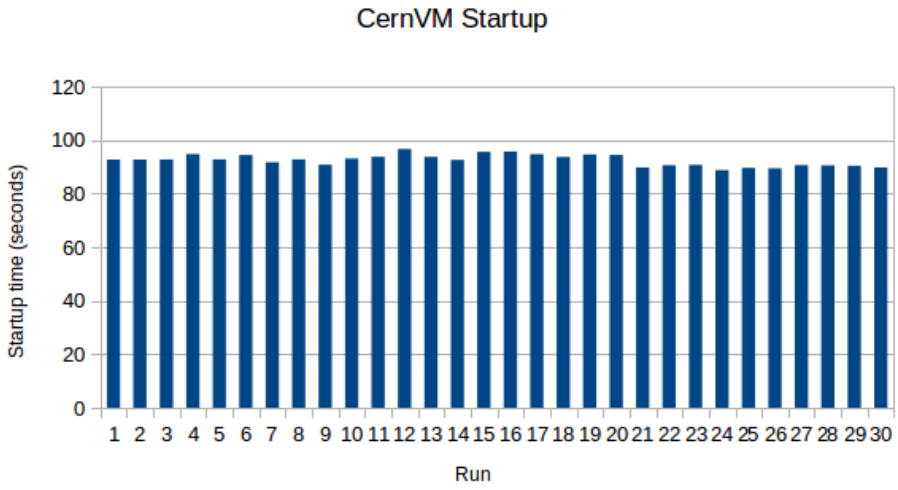

Figure 8.5: Starting instances on the testbed on average take 92.6 seconds with a minimum of 88.8 seconds and a maximum of 96.7 seconds

be considered a waste of time.

8.6.3 Job Execution

Some benchmarking of CernVM has already been done[56] on different hypervisors, including KVM, and have been compared to a native linux installation. The applications used for testing are typical Grid analysis software and as OpenStack use the KVM as hypervisor by default, these benchmarks should still be relevant. The referred study was done in 2010/2011 by Brynjulv Brynjulvsen.

From the figures 8.6 and 8.7 we see that KVM was not the best suited hypervisor for CernVM running experiment software. However, more recent benchmarks (2013)[57] of KVM matched against other hypervisors have been done. A study done at the George Washington University along with IBM have compared the performance of KVM, vSphere, VMWare and XEN. These tests have not used CernVM or any typical analysis software, but test how the hypervisors perform under different tests. The study done at the George Washington University conclude that the hypervisors outperform each other within different areas, and that there is no single hypervisor outperforming the other altogether. It also suggest that matching hypervisors to different applications require more attention.

With the application performance measured, the most interesting part of the job execution for this project is the downloading of AliEn and the required software, using CVMFS, to start a job agent. This process was measured by recording the time before starting the job agent and after retrieving a short job (executing /bin/date).

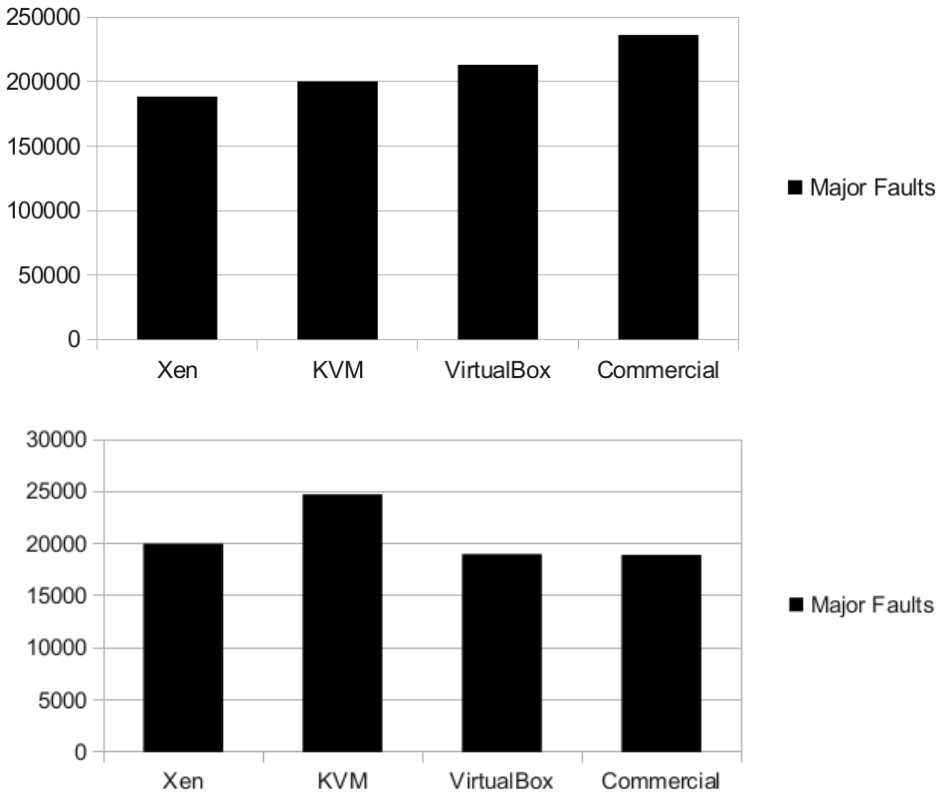

Figure 8.6: CernVM page faults running ppBench with 512MB (upper graph) and 1024MB (lower graph) RAM[56]

The results were then compared with a cached CVMFS node to be able to extract the download time.

Figure 8.8 displays the download times of the software required to start an AliEn job agent from the job agent startup script of the EC2 user data. The measurements were done at the network of Bergen University College downloading from the repositories of ALICE. The amount of data to be downloaded to start a job agent is 39.5MB and with the average of 33.5 seconds the average download speed from the repositories are about 1.17MB/s or ~9.4 Mbps with a peak of ~15.8 Mbps and a minimum of ~7 Mbps. The slow download rates are clearly a drawback and could be greatly increased by using a local cache on the site. By having the disposable worker nodes use a local cache, the time consumed by downloading AliEn software can in theory be decreased to ~3.16 seconds on a 100Mbps LAN and obviously even further decreased on a 1Gbps LAN.

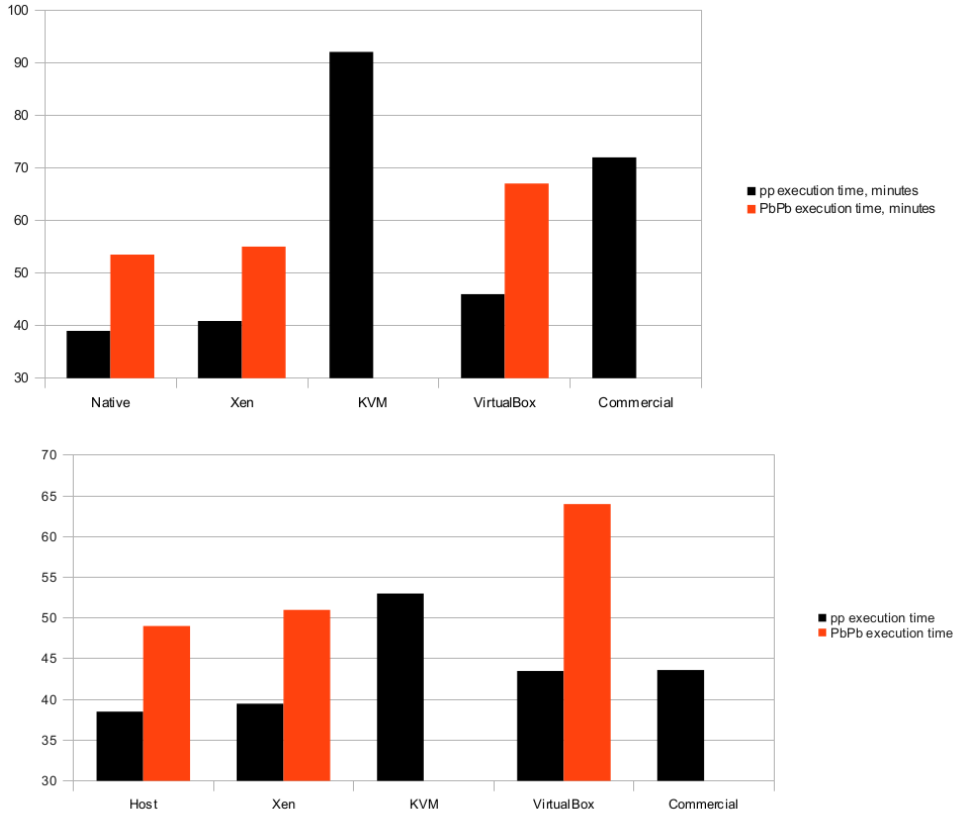

Figure 8.7: CernVM execution time of different simulation software with 512MB (upper graph) and 1024MB (lower graph) RAM[56]

8.6.4 Lifecycle management

To test the lifecycle management several short jobs was submitted. Short jobs gives the virtual worker node a short lifetime, so that the spawning and killing of the virtual worker nodes can easily be observed in the AliEC2 logs or from the output of AliEC2. By submitting more jobs than the system can provide virtual machines for, the limiting of virtual machines can also be tested. The VM limit is a configurable size dictating how many VM instances the system can handle. To test the limiting, virtual machines with fewer resources increase the VM limit which the system can provide.

From observing the logs of AliEC2, the functionality of the lifecycle management service is working correctly on the testbed specified in chapter 7. New virtual worker nodes are spawned when AliEn::LQ::EC2 request new worker nodes, provided that the system have free resources. When there are no more jobs for the job agents on the virtual worker nodes to do, the worker nodes tell AliEC2 that they are done, followed by AliEC2 deleting them.

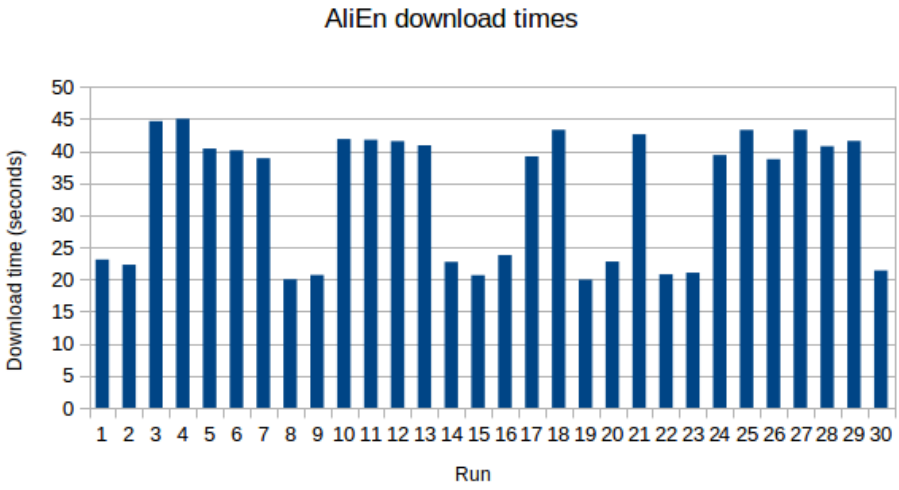

Figure 8.8: The required software for starting the job agent is downloaded by CVMFS. The amount of data to download is 39.5MB and on average this process takes 33.5 seconds.

8.6.5 Summary

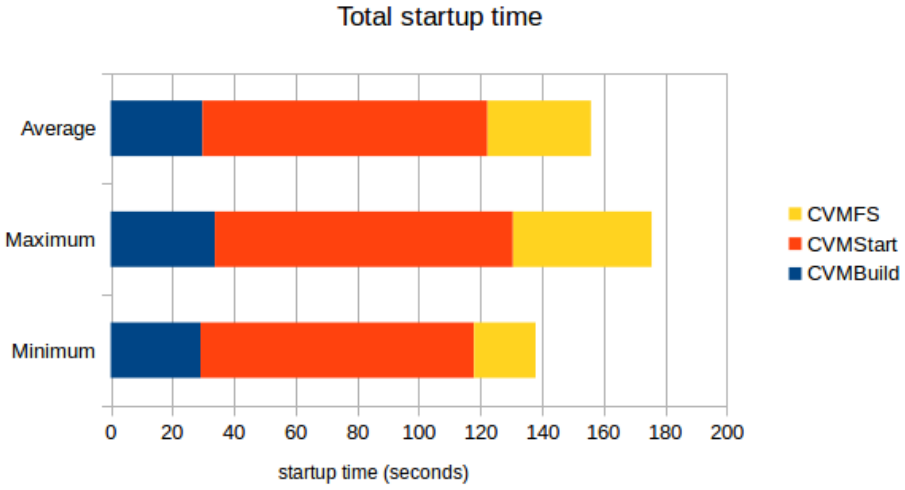

Figure 8.9: The total time consumed before the job agent starts may vary from ~137 seconds up to ~175 seconds.

By adding the time consumed in the steps described in the above sections we get an estimate of the total overhead of the virtual machines. Figure 8.9 shows that in total the solution on this testbed give an average job agent startup time of ~155 seconds, or close to 2:30 minutes. A job agent normally have a maximum lifetime of 48 hours. An extra startup time of two minutes and thirty seconds are therefore quite low.

The startup time could be optimized in a few ways, but the most drastic change could be done by simply adding a local cache to the site as seen in section 8.6.3, which could reduce the startup time by 20-30 seconds. Moreover, by looking at the logs from the CernVM startup (section 8.6.2), some optimizations of the CernVM startup process could be done to shorten the startup process of the CernVM instances, e.g. disabling the file system check.

Chapter 9

Similar Solutions

Some projects working towards a similar goal exist. Although they are somewhat similar, there's still some important differences. This chapter will give a brief introduction to some of these projects, describe the key differences from our project, and the purposes of these projects versus this proposed solution.

9.1 Cloud Scheduler

Cloud Scheduler[58] is developed at the University of Victoria, Canada by the High Energy Physics Research Computing group (HEPRC), along with the CANFAR project, and NRC-Sussex in Ottawa. It is designed for use in the CANFAR project[59] and in the HEP Legacy Data Project[60], both of which are funded by CANARIE[61].

From figure 9.1 we observe that the scheduler looks at the job-queue to discover which VM-Image to boot up for a job. This image must be specified in the job submission file and it must also be prepared and uploaded by the user. In addition the user must specify other VM parameters like CPU cores, memory, CPU architecture, network type and storage.

This goes against one principle of our proposed solution; the user of the Grid should have no concern regarding where and on which platform the job is running. Cloud scheduler is also designed to be running as a complete solution for its targeted projects, as opposed to our solution which is designed as an additional component to an already existing system (AliEn).

9.2 CoPilot

CoPilot[62] is a project at CERN developed by the CernVM team. From figure 9.2 we see the components of CoPilot: the job manager, agent, storage and monitor

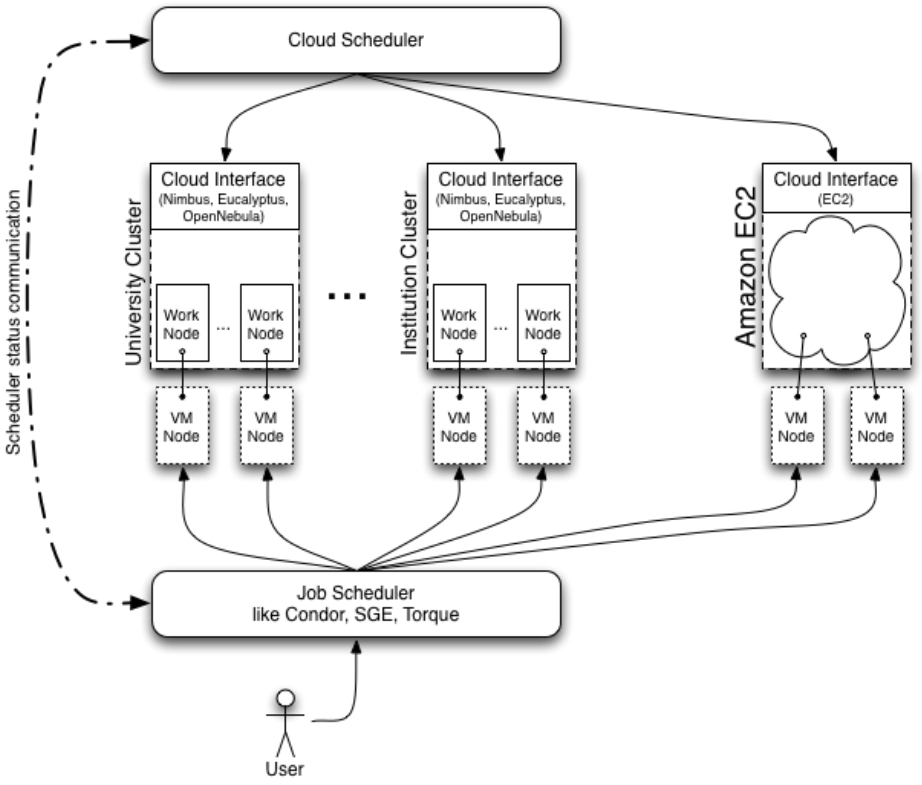

Figure 9.1: An overview of the Cloud Scheduler architecture.[58]

services, interfacing different Grid systems with cloud systems. The job manager maintains the job queue, keeps track of job states and distribute jobs for execution. The agents request jobs from the job manager, execute the jobs and upload the job results to the storage. The monitor collects and stores information about the state of the system, data such as number of jobs in the queue, and the space available in the storage. New resources running the CoPilot agent register to the CoPilot Manager.

CoPilot is designed to collect and orchestrate different forms of free resources from volunteer PCs to Cloud Instances. The resources are not requested based on demand, and the resources are not freed after a job is done executing, lacking the lifecycle management needed by the problem described in section 1.4, and does not provide elasticity on the same level as our solution.

9.3 ROCED

ROCED (Responsive on-demand Cloud Enabled Deployment)[63] is a meta-scheduler developed by researchers from the Karlsruhe Institute of Technology. This solution

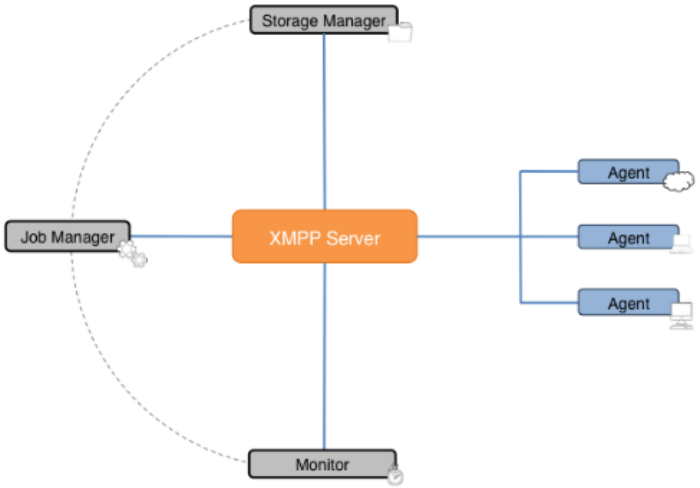

Figure 9.2: The architecture of CernVM-CoPilot[62]

require a customized OS image with Torque or PBS installed running in the cloud and is primarily designed to extend static resources on local clusters by using public cloud resources.

9.4 Summary

Cloud scheduler, CoPilot and ROCED are more general purpose and in some cases are replacements for AliEn on the clusters, whereas our solution is an add-on to the existing AliEn cluster setup. Moreover, some solutions are not transparent to the users of the Grid, and some solutions lack the key functionality this prototype provide. Another noticeable difference is that all of them need batch systems, whereas our solution use cloud functionality to replace the needs for batch systems.

Chapter 10

Evaluation and Conclusion

There are several factors to evaluate, regarding the requirements described in section 8.1 and issues concerning possible loss of performance. Evaluating the requirements is a matter of matching the solution with the requirements and discussing whether the solution fulfills the specified requirements.

10.1 Evaluation of requirements

This section will focus on the requirements of the solution, beginning with a short recap on the requirements. The solution should

- not run any virtual worker nodes while there are no jobs to run.
- have a minimal footprint in the existing AliEn grid.

The first requirement is fulfilled by implementing a service managing the lifecycle of the virtual machine instances, including starting and stopping VM instances, as well as taking care of any errors which might occur to the instances. The reporting system facilitates detecting unresponsiveness from a machine, and non responsive virtual machines will be deleted after a limited amount of time.

The second requirement is a bit harder to evaluate as the footprint is hard to measure. Installing this extension on a Grid site should be simple as guides and some scripts are provided along with this thesis (see chapter 7 and appendix A). As mentioned in section 8.1, the amount of work to be done while applying this proposed solution may vary from site to site depending on the software already installed on the site.

Setting up AliEn and OpenStack correctly is comprehensive work. The interface and service developed in chapter 8 is simply a matter of installing the required perl

libraries and frameworks described in chapter 7 followed by setting AliEn to use the AliEn::LQ::EC2 interface and starting the AliEC2 service.

10.2 Performance Evaluation

Evaulating the performance is a matter of what amount of overhead is acceptable. This value is not specified anywhere, leaving this report to reason about what is acceptable and what is not.

10.2.1 CernVM performance

Some testing have already been done as mentioned in section 8.6.3. One of these tests was conducted in 2010/2011 comparing different hypervisors running CernVM with different physics analysis software. This study concluded that KVM was not the most suited hypervisor for hosting CernVM instances. However, the second study conducted in 2013 which compared different hypervisors under different circumstances, including KVM, concluded that none of the hypervisors tested outperformed each other. These two tests tell that the KVM performance might begin to challenge other hypervisors, but also that the overhead of all the hypervisors should be acceptable for AliEn jobs.

10.2.2 AliEC2 Performance

The performance benchmarking done in section 8.6 shows that the total job execution time will have an average additional time of ~two minutes and thirtyfive seconds, varying from ~two minutes and seventeen seconds to ~two minutes and fiftyfive seconds. Keeping in mind that the maximum lifetime of a Job Agent normally is set to 48 hours, the additional VM startup times are very low. The results from section 10.2.1 show that in 2011 the CernVM performance along with KVM was not the best, but later benchmarking suggest that in more general cases KVM perform similarly well as other hypervisors.

The lifecycle management service has not been tested on more than 12 concurrent instances, but the tests show that the service is performing as it should; starting when requested and deleting instances when they are done or have not updated, and also ignoring new instance requests when the instance limit has been reached.

10.3 Further Work

The implementation of the proposed solution is a functioning prototype. Much have been done, but some issues remains.

This solution has not taken any security aspects into consideration. Authentication could be added to the web service component of AliEC2 to improve the security.

Investigation of the overhead may further reduce the startup time. One approach for reducing the overhead might be to always have a number of extra VM instances active, ready to start a job agent. When a new job is submitted, the active instance grab the job while a new instance is started. Moreover, the overhead of the current implementation could be reduced by adding a local CVMFS-cache to the site. Additionally, the CernVM startup process could be better fitted for this prototype, e.g. the startup time could be shortened by removing the file system check as mentioned in section 8.6.

10.4 Conclusion

CernVM along with OpenStack, and possibly other EC2 enabled cloud systems, seems well suited for providing an elastic and homogenous Grid environment. The extra startup time of less than three minutes is negligible compared to a job agent lifecycle of 48 hours. The complexity of applying the implementation on a data center are mostly dependent on whether the center use an EC2 enabled cloud system or not. With OpenStack and AliEn already correctly set up, applying the implementation should be fairly trivial with the resources provided in appendix A and in chapter 7.

Appendices

Appendix A

AliEC2 Installation

Install Perl libraries and frameworks required by AliEC2 and EC2.pm from CPAN:
sudo cpan YAML DBI DBD::SQLite Net::Curl::Easy Net::Curl::Form Dancer Dancer::Logger::Log4perl Log::Log4perl DateTime DateTime::Format::DBI DateTime::Format::SQLite VM::EC2 Config::Simple

The installation script for setting up OpenStack all-in-one can be downloaded from github:
git clone https://github.com/Joachricar/Joastack.git
The installer can be run by ./install.sh -g to generate the answer-file. After checking that all values are correct, run ./install -a to install with the packstack-answers.txt generated. This script installs OpenStack all-in-one, creates a security group, adds keypair for the user executing the installer, downloads CernVM 2.7.1 batch node from cernvm.cern.ch, creates a CernVM qcow2 volume and starts an instance of CernVM.

AliEC2 can also be downloaded from github:
git clone https://github.com/Joachricar/AliEC2.git
The file aliec2ws.pl is the executable for AliEC2. ec2.conf is used to configure the service.

Bibliography

[1] ALICE website. http://aliceinfo.cern.ch. Accessed: 17-02-2014.

[2] About CERN. http://home.web.cern.ch/about. Accessed: 2014-03-03.

[3] Barry Wilkinson. *Grid Computing: Techniques and Applications*. Chapman and Hall, 2009.

[4] P. Saiz, L. Aphecetche, P. Bunčić, R. Piskač, J.-E. Revsbech, and V. Šego. Alien—alice environment on the {GRID}. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 502(2–3):437 – 440, 2003. Proceedings of the {VIII} International Workshop on Advanced Computing and Analysis Techniques in Physics Research.

[5] Cloud computing wikipedia. http://en.wikipedia.org/wiki/Cloud_computing. Accessed : 19-05-2014.

[6] Openstack.org. http://www.openstack.org/. Accessed: 2013-05-14.

[7] Predrag Buncic, Carlos Aguado-Sanchez, Jakob Blomer, and Artem Harutyunyan. CernVM: Minimal maintenance approach to virtualization. *Journal of Physics: Conference Series*, 331(5):052004, 2011.

[8] Grid computing. https://en.wikipedia.org/wiki/Grid_computing. Accessed: 2013-05-09.

[9] Grid middleware. http://en.wikipedia.org/wiki/List_of_grid_computing_middleware_distribution. Accessed: 2013-05-02.

[10] Alice environment. http://alien2.cern.ch/. Accessed: 2013-05-09.

[11] About ACE. http://home.web.cern.ch/about/experiments/ace. Accessed: 2014-03-03.

[12] About UA9. http://home.web.cern.ch/about/experiments/ua9. Accessed: 2014-03-03.

[13] About ALICE. http://home.web.cern.ch/about/experiments/alice. Accessed: 2014-03-03.

[14] Beowulf cluster. http://en.wikipedia.org/wiki/Beowulf_cluster. Accessed: 2014-30-04.

[15] T.C. Wilcox. *Dynamic Load Balancing of Virtual Machines Hosted on Xen.* Brigham Young University. Department of Computer Science, 2009.

[16] O-Code Wikipedia. `http://en.wikipedia.org/wiki/O-Code`. Accessed: 2014-03-03.

[17] BCPL Wikipedia. `http://en.wikipedia.org/wiki/BCPL`. Accessed: 2014-03-03.

[18] JVM Wikipedia. `http://en.wikipedia.org/wiki/JVM`. Accessed: 2014-03-03.

[19] Edouard Bugnion, Scott Devine, Kinshuk Govil, and Mendel Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. *ACM Trans. Comput. Syst.*, 15(4):412–447, November 1997.

[20] A brief history of cloud computing. `http://blog.softlayer.com/2013/virtual-magic-the-cloud`. Accessed: 2014-03-03.

[21] Public Key Infrastructure Wikipedia. `http://en.wikipedia.org/wiki/Public_key_infrastructure`. Accessed: 2014-22-05.

[22] Job Submission Description Language. `http://en.wikipedia.org/wiki/Job_Submission_Description_Language`. Accessed: 2014-21-03.

[23] AliEn jobs. `http://alien2.cern.ch/index.php?option=com_content&view=article&id=52&Itemid=100`. Accessed: 2014-21-03.

[24] ClassAd. `http://research.cs.wisc.edu/htcondor/classad/`. Accessed: 2014-24-05.

[25] Junwei Cao, Daniel P. Spooner, Stephen A. Jarvis, and Graham R. Nudd. Grid load balancing using intelligent agents. *FUTURE GENERATION COMPUTER SYSTEMS*, 21(1):135–149, 2005.

[26] MonALISA AliEn. `http://alimonitor.cern.ch`. Accessed: 2014-24-03.

[27] AliEn Status March 2014. `http://indico.cern.ch/event/305441/session/5/contribution/36/material/slides/1.pdf`. Accessed: 2014-21-03.

[28] Philippe Gros, AndersRhod Gregersen, Jonas Lindemann, Pablo Saiz, and Andrey Zarochentsev. Interoperating AliEn and ARC for a Distributed Tier1 in the Nordic Countries. In Simon C. Lin and Eric Yen, editors, *Data Driven e-Science*, pages 201–210. Springer New York, 2011.

[29] TORQUE Resource Manager. `http://www.adaptivecomputing.com/products/open-source/torque/`. Accessed: 2014-23-05.

[30] HTCondor web site. `http://research.cs.wisc.edu/htcondor/`. Accessed: 2014-23-05.

[31] Alvise Dorigo, Peter Elmer, Fabrizio Furano, and Andrew Hanushevsky. XROOTD/TXNetFile: A Highly Scalable Architecture for Data Access in the ROOT Environment. In *Proceedings of the 4th WSEAS International*

Conference on Telecommunications and Informatics, TELE-INFO'05, pages 46:1–46:6, Stevens Point, Wisconsin, USA, 2005. World Scientific and Engineering Academy and Society (WSEAS).

[32] S Bagnasco, L Betev, P Buncic, F Carminati, C Cirstoiu, C Grigoras, A Hayrapetyan, A Harutyunyan, A J Peters, and P Saiz. AliEn: ALICE environment on the GRID. *Journal of Physics: Conference Series*, 119(6):062012, 2008.

[33] Iosif Legrand, Harvey Newman, Ramiro Voicu, Costin Grigoras, Catalin Cirstoiu, et al. MonALISA: A distributed service system for monitoring, control and global optimization. *PoS*, ACAT08:020, 2008.

[34] Peter Mell and Timothy Grance. The nist definition of cloud computing. Technical Report 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD, September 2011.

[35] K Fransham, A Agarwal, P Armstrong, A Bishop, A Charbonneau, R Desmarais, N Hill, I Gable, S Gaudet, S Goliath, R Impey, C Leavett-Brown, J Ouellette, M Paterson, C Pritchet, D Penfold-Brown, W Podaima, D Schade, and R J Sobie. Research computing in a distributed cloud environment. *Journal of Physics: Conference Series*, 256(1):012003, 2010.

[36] Synergy research. Cloud provider revenue. `https://www.srgresearch.com/articles/ibm-microsoft-and-google-still-make-little-headway-q3-against-amazons-iaaspaas` Accessed: 25-02-2014.

[37] Opennebula website. `http://opennebula.org/`. Accessed: 2014-27-02.

[38] Apache cloud stack. `http://cloudstack.apache.org/`. Accessed: 2013-05-14.

[39] Eucalyptus website. `https://www.eucalyptus.com/`. Accessed: 2014-27-02.

[40] NASA Website. `http://www.nasa.gov/`. Accessed: 2014-27-02.

[41] Rackspace website. `http://www.rackspace.com/`. Accessed: 2014-27-02.

[42] About openstack. `https://wiki.openstack.org/wiki/Main_Page`. Accessed: 2013-27-02.

[43] Openstack operator training guide. `http://docs.openstack.org/training-guides/content/operator-getting-started.html`. Accessed: 2014-19-03.

[44] Removal of shutdown behavior. `https://lists.launchpad.net/openstack/msg12640.html`. Accessed: 2014-19-03.

[45] Understanding full virtualization, paravirtualization, and hardware assist vmware white paper. `http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf`. Accessed: 2014-24-03.

[46] Jeos. http://en.wikipedia.org/wiki/Just_enough_operating_system. Accessed: 2014-10-03.

[47] P. Buncic, C. Aguado Sánchez, J. Blomer, A. Harutyunyan, and M. Mudrinic. A practical approach to virtualization in HEP. *The European Physical Journal Plus*, 126(1), 2011.

[48] CernVM website. http://cernvm.cern.ch. Accessed: 2014-24-03.

[49] QEMU Copy on Write. http://en.wikipedia.org/wiki/Qcow. Accessed: 2014-14-05.

[50] Eucalyptus command line interface. https://www.eucalyptus.com/download/euca2ools. Accessed: 2014-24-05.

[51] LibVM EC2 Perl. https://github.com/lstein/LibVM-EC2-Perl. Accessed: 2014-19-03.

[52] GitHub code edit request. https://github.com/lstein/LibVM-EC2-Perl/issues/15.

[53] Perl Dancer website. http://perldancer.org. Accessed: 2014-27-04.

[54] Comprehensive Perl Archive Network. http://www.cpan.org/. Accessed: 2014-27-04.

[55] Apache Directory Studio website. https://directory.apache.org/studio/. Accessed: 2014-27-04.

[56] Brynjulv Mathias Brynjulvsen. Virtual machines in computational grids. Master's thesis, Bergen University College.

[57] Jinho Hwang, Sai Zeng, Frederick Wu, and Timothy Wood. A component-based performance comparison of four hypervisors. In Filip De Turck, Yixin Diao, Choong Seon Hong, Deep Medhi, and Ramin Sadre, editors, *IM*, pages 269–276. IEEE, 2013.

[58] Cloud scheduler. http://cloudscheduler.org/. Accessed: 2014-27-02.

[59] Canfar project. http://www.canfar.phys.uvic.ca/canfar/about.html. Accessed: 2014-27-02.

[60] Hep legacy data project. http://heprc.phys.uvic.ca/legacyproject/. Accessed: 2014-27-02.

[61] Canarie network enabled projects. http://www.canarie.ca/en/network-programs/network-platforms/nep/projects. Accessed: 2014-27-02.

[62] A Harutyunyan, J Blomer, P Buncic, I Charalampidis, F Grey, A Karneyeu, D Larsen, D Lombraña González, J Lisec, B Segal, and P Skands. CernVM Co-Pilot: an Extensible Framework for Building Scalable Computing Infrastructures on the Cloud. *Journal of Physics: Conference Series*, 396(3):032054, 2012.

[63] T Hauth, G Quast, M Kunze, V Büge, A Scheurer, and C Baun. Dynamic extensions of batch systems with cloud resources. *Journal of Physics: Conference Series*, 331(6):062034, 2011.