Facultad de Ciencias

Dinamización de cargas de trabajo HPC/HTC: conciliando modelos "on-premise" y "cloud computing"
(Job workload dynamization: reconciling on-premise and cloud-computing models)

Trabajo de Fin de Máster
para acceder al

MÁSTER EN INGENIERÍA INFORMÁTICA

Autor: Pedro Andrés Heredia Canales

Director\es: Jose Ángel Herrero Velasco

Octubre - 2020

Contents

List of Figures

Acknowledgements

This thesis would not have been possible without the inspiration and support of a number of wonderful individuals — my thanks and appreciation to all of them for being part of this journey and making this thesis possible.

First and foremost, I would like to thank my family for their continuous love, help and support. I am greatly indebted to my parents for giving me the opportunity to study what I like even when that meant living more austerely. Your task is now completed and it is now my duty to take care of you. I am grateful to my girlfriend Andreea Daiana Tipi who is already part of the family, for always being a great support as well as being the light on my darkest days.

I would also like to thank my thesis supervisor Jose Ángel Herrero Velasco that with his infinite patience and optimism guided and supported me throughout the duration of this thesis. Thanks also to Pablo Abad Fidalgo for his invaluable help in the last steps of this thesis.

Finally, I would like to thank all the colleague and teachers I had in the master for creating a relaxed and collegial atmosphere in class, making the learning process really enjoyable while keeping it challenging and interesting.

Resumen

Con el enorme crecimiento que ha experimentado la computación en la nube durante la última década, evolucionando desde un concepto tecnológico a ser considerada un modelo de negocio completo, las entidades que demandan recursos computacionales se empiezan a plantear la migración completa de estos recursos a la nube. Sin embargo, seguir esta tendencia de tener todo en la nube puede no ser necesariamente la mejor opción y, quizás, como en muchas otras cosas, la respuesta esté en algo intermedio.

Si bien actualmente las principales compañías que gestionan los grandes entornos *cloud* comerciales, como son Google, Microsoft o Amazon, están llevando a cabo grandes esfuerzos en el desarrollo e implementación de la llamada nube híbrida, estos concentran principalmente su atención en la evolución de plataformas para desarrollo de software, como ocurre por ejemplo en el caso de Microsoft Azure, con su producto Azure Stack, destinado al desarrollo y ejecución de aplicaciones híbridas (que pueden ejecutar tanto *on-premise* como en *cloud*), mientras que los entornos para la ejecución de aplicaciones HPC/HTC, parece que quedan relegados a un segundo plano. Su baja demanda actual viene motivada por diversas causas, entre las que caben destacar la necesidad de hardware muy específico (en muchas ocasiones de muy altas prestaciones), los problemas derivados del *overhead* creado por la virtualización, y no menos importante, el factor económico, ya que este tipo de infraestructuras personalizadas pueden alcanzar un precio mucho mayor que las de configuración más estándar.

Sabiendo esto y teniendo en cuenta que, en la inmensa mayoría de casos, migrar completamente a la nube puede ser desaconsejable debido entre otras cosas, a la existencia previa de una infraestructura local que provee de los recursos necesarios, en cuanto a computación y almacenamiento se refiere, este trabajo pretende explorar una solución a medio camino entre la computación *on-premise* y la computación en la nube, o también llamada *cloud-computing*, que permita a un usuario HPC/HTC beneficiarse de la computación en la nube sin prescindir del entorno *on-premise* al que está acostumbrado, siendo capaz de ejecutar trabajos en ambos entornos.

Para ello, se ha desarrollado el framework *Hybrid-Infrastructure-as-a-Service Manager* (HIaaS-M), que pretende conciliar los dos paradigmas, automatizando la interacción entre ellos de forma eficiente y completamente transparente para el usuario.

Este framework está especialmente diseñado para su integración en infraestructuras ya existentes (on-premise) de forma también transparente, es decir, sin necesidad de modificar ninguna pieza software. Su ejecución se realiza de manera independiente, como un programa autónomo, que se comunica con los sistemas existentes, minimizando así el impacto que puedan suponer posibles cambios en las piezas software que componen la infraestructura donde se vaya a implantar.

A lo largo de esta memoria se describe el proceso completo de desarrollo de este framework, modular y configurable, el cual permite la integración de una infraestructura computacional existente con la proporcionada por un entorno cloud, añadiendo la posibilidad de ejecutar trabajos en prácticamente cualquiera de los entornos cloud apoyándose fundamentalmente en el uso de la librería Libcloud.

El trabajo culmina con una prueba de concepto realizada sobre el cluster en desarrollo (de nombre cluster2) ubicado en el CPD 3Mares de la Facultad de Ciencias de la Universidad de Cantabria. Este último paso nos ha permitido concluir el trabajo identificando las ventajas del framework así como algunas consideraciones a tener en cuenta para trabajos futuros.

Palabras clave: IaaS, Cloud, HPC, HTC, Manager, On-premise, Hybrid

Abstract

Cloud computing has grown tremendously in the last decade, evolving from being a mere technological concept to be considered a full business model. Some entities like companies or research groups, that have a need for computational power are beginning to consider a full migration of their systems to the cloud. However, following the trend of full migration to the cloud might not be the optimal option, perhaps, not everything is black and white and the answer could be found somewhere in between.

Although great efforts are being made in the development and implementation of the so called hybrid cloud by companies that manage the biggest commercial cloud environments, namely Google, Amazon and Microsoft, most of them are focused in the creation of software developing platforms like in the case of Azure Stack from Microsoft Azure, that helps to develop hybrid applications that can be executed both locally and in the cloud. Meanwhile, the provisioning of execution environments for HPC/HTC applications seems that to be relegated to the background. In part, this could be because currently there is a low demand for these environments. This low demand can be motivated by many factors among which it is worth highlighting the necessity of having really specialised hardware, the overhead introduced by virtualization and last, but not least, the economic impact usually associated to this kind of customized infrastructures in contrast with more standard ones.

With these limitations in mind and the fact that, in most of the cases, complete migration to the cloud is limited by the previous existence of a local infrastructure that provides computing and storage resources, this thesis explores an intermediate path between on-premise (local) and cloud computing. This kind of solution will allow an HPC/HTC user to benefit from the cloud schema in a transparent way, maintaining the on-premise environment that he is so used with, and also being able to execute jobs in both paradigms.

To achieve this, the Hybrid-Infrastructure-as-a-Service Manager (HIaaS-M) framework is created. This framework tries to join both computing paradigms by automating the interaction between them in a way that is efficient and completely transparent to the user.

This framework is especially design to be integrated into already existing infrastructures (on-premise); in other words, without the need of changing any of the existing software pieces. The framework is a standalone software that communicates with the existing systems, minimizing the impact that changing the base software and/or infrastructure of an entity could cause.

This document describes the whole development process of this modular and configurable framework which allows the integration of previously existing infrastructures with one created in the cloud through a cloud infrastructure provider, adding the alternative of executing jobs in any of the cloud environments provided by cloud providers thanks to the Apache Libcloud library.

This document concludes with a proof of concept made over a development cluster (called ”cluster2”) hosted in the 3MARES Data Processing Center at the Science Faculty of the University of Cantabria. This deployment on a similar to real life environment has allowed to identify the main advantages of the framework, as well as improvement that could be made that are expressed in a suggested roadmap for future work.

Keywords: IaaS, Cloud, HPC, HTC, Manager, On-premise, Hybrid

Chapter 1

Introduction

The use of computational resources has become an essential need in most of the fields. That is why, it is fairly safe to assume that every modern entity like companies, research groups, etc., make use of some form of computational resources.

The requirements of these computational resources can vary widely as they depend on the intended usage. While a small company could satisfy its needs with a ”basic” infrastructure, big companies (or big research groups, governments, etc.) might need a more sophisticated and specialized infrastructure to cover their needs.

Until not long ago, the only option to have access to these resources was to buy them. It was not until the year 2006, when big companies like Google or Amazon started using the term *cloud computing*, that a new option appeared. Now it was possible to ”rent” computational resources through cloud providers like Amazon AWS and, later on, Google Cloud and Microsoft Azure.

Whether bought or rented, computational resources can have an important economic impact, especially in big entities that require a lot of computational power. This impact is usually measured by using the term Total Cost of Ownership (TCO) [37] that takes in consideration not only the purchase cost but the operational cost as well. The operational cost of computational resources may include: software licensing cost, hardware maintenance, utility charges (electrical power and cooling systems) or the required skills of the system administrators (higher skills means higher salaries). As a result, different entities could have different TCOs having the same computational needs.

Keeping this in mind, each entity has to individually analyse which one of the 3 available paradigms cover their computational needs while being economically viable for them.

- The first option is to buy the needed resources and to placed them in their own facilities, creating their own computing infrastructure, usually called on-premise infrastructure or private cloud.

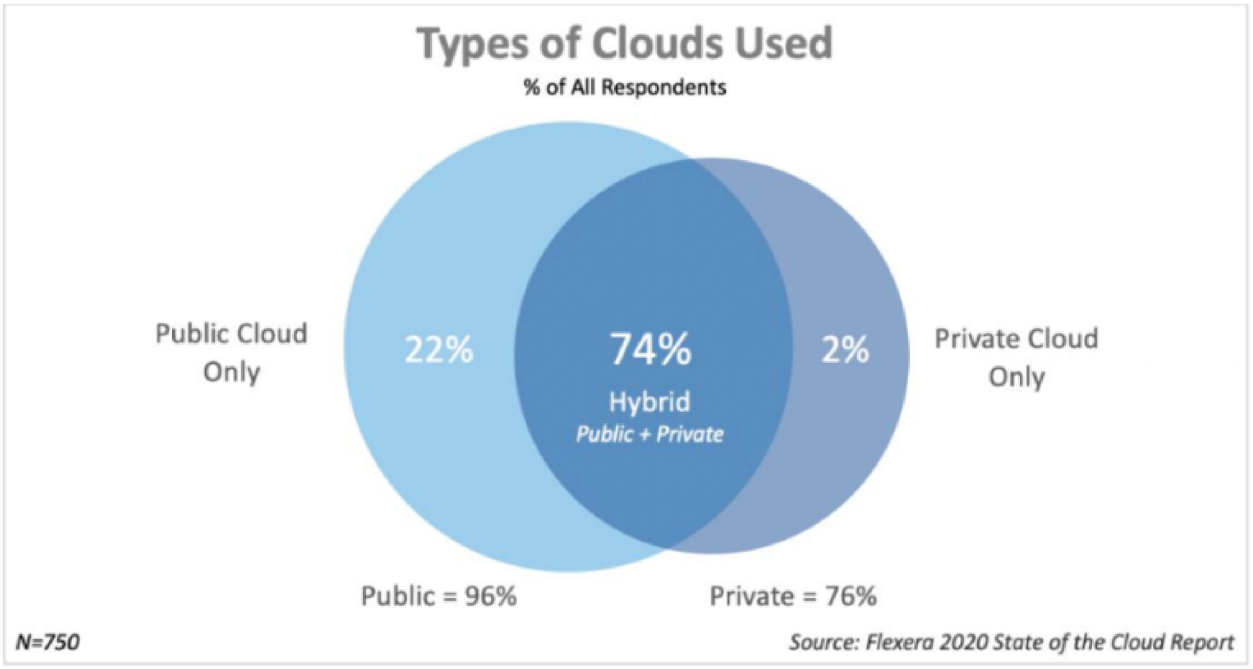


Figure 1.1: Cloud types of users [11]

- A second option is to "rent" them through a cloud provider, not needing facilities or workforce and paying only for what they use. This is usually called cloud computing.
- A third option it to have a mix of both, on-premise and cloud computing infrastructures. This is usually called hybrid cloud.

It seems clear that this is not a trivial decision; therefore, currently there are mixed opinions about which of the paradigms is the optimal one.

Even though in the last years there has been a huge marketing push in favour of the cloud computing paradigm, independent reports like The Flexera 2020 State of the Cloud Report [8] gives some insights on the trends in the cloud business. As figure 1.1 shows, only 22% percent of users are using the cloud computing paradigm exclusively. On the other hand, 74% of the users are using some kind of hybrid computing. In other words, they have both on-premise (private cloud) and cloud infrastructures.

There are many reasons why entities might not be choosing the cloud computing paradigm as their only option and, even most of them have to internally and individually evaluated by each entity, some of these reasons can be objectively discussed.

There are entities in which their work or research fields are focused on solving complex problems with big quantities of data that require large computational resources. To solve these problems the computing power of a single machine is often not enough. Therefore, the way to tackle them is to use aggregate computing or supercomputing. As [36] explains, the term ”supercomputing” refers to the processing of massively complex or data-laden problems using the concentrated compute resources of multiple computer systems working in parallel (i.e. a ”supercomputer”).

Inside the supercomputing field, there are two main branches High Performance Computing and High Throughput Computing (HPC/HTC). Both branches allow scientists and engineers solve complex problems that usually involves a large amount of data and a high demand for computing power. Infrastructure wise, both HPC and HTC applications demands high level of computational resources, including high performance networks and high performance storage systems.

There is, however, a key difference between the two paradigms HPC and HTC: the capacity of parallel execution.

In HPC, the applications can be executed in clusters, meaning they will execute in several computing environments each one of them called a node. This nodes are interconnected to each other using high performance networks (high bandwidth, very low latency) so that an application can be divided in ”n” instances or processes that are distributed through the nodes, where they will execute in a controlled and coordinated manner to ultimately solve the problem it was designed for. In this distributed computing model, all the elements have high performance profiles with the goal of executing this applications in the less amount of time possible.

On the other hand, there is HTC, which main goal is to create an environment focused on finishing the largest amount of tasks (jobs, tasks, applications) in a certain amount of time. This requires not only using the available resources in an efficient way, but also to have as many resources as possible. The applications designed to use this paradigm usually do not need high performance networks but they do need high computing power.

Reviewing all that has been previously said, entities that use or want to use paradigms like HPC/HTC could have the same problem deciding which one of the three computing infrastructure paradigm to use.

Traditionally, HPC/HTC was almost exclusive to on-premise computing in part because it offers total control over the machines and minimize the performance losses. However, studies are beginning to place in doubt this. As mentioned in [10] and [18] the main concern, cost apart, about HPC/HTC applications executing on the cloud is the performance output compared to the one obtained in on-premise infrastructures, especially when dealing with algorithms that highly depend on communication. Much has changed since these studies and the difference in performance has narrowed since then, partially because this issue is being addressing by the main cloud providers by offering more and more customized machines, to the point where the client is able to pay for bare metal machines with tailored network systems. Nonetheless, the economic factor still remains as bare metal solutions are significantly more expensive than the standard ones.

For these entities, the third option (to use a hybrid cloud paradigm) could be a really interesting and

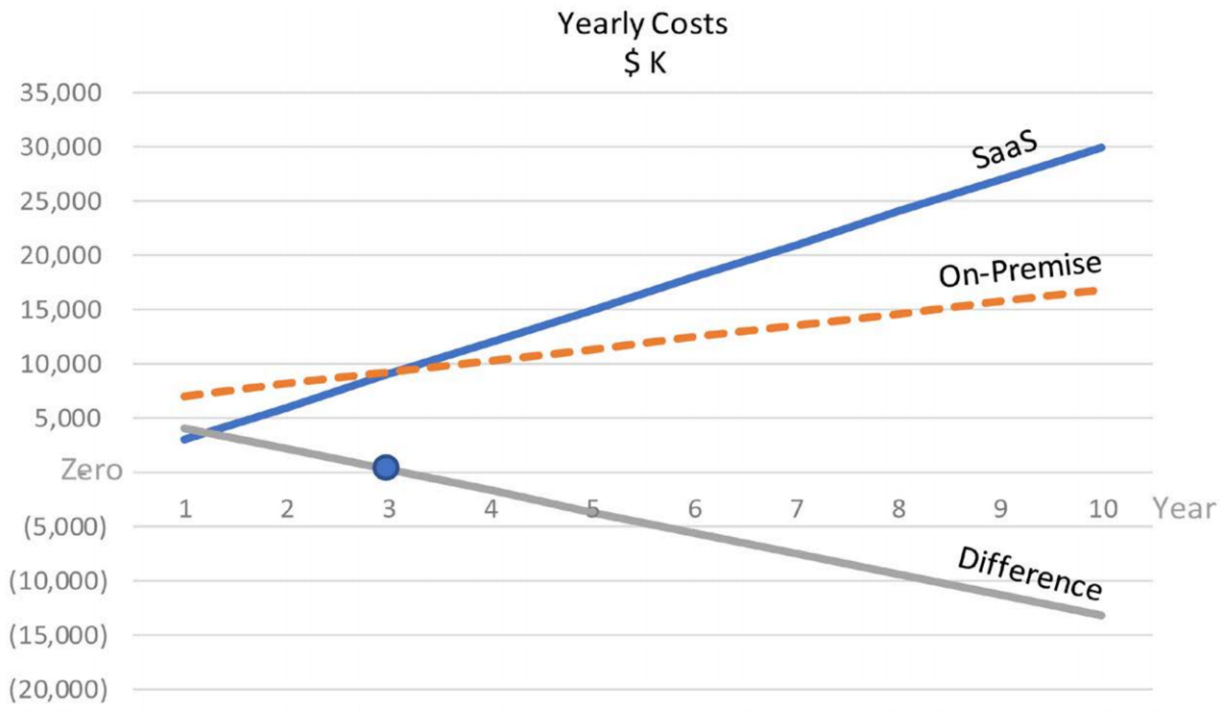


Figure 1.2: Yearly cost of using Cloud vs On-premise [10]

suitable option as it takes advantage of both of the models. This statement is reinforced by the numbers seen in figure 1.1, that show the hybrid cloud as the most common paradigm. However, using this paradigm often implies a high cost, not only economically speaking, but also in terms of the time and the effort needed to adapt the existing on-premise infrastructure to work with a cloud environment.

Summarizing this introduction section, it could be said that the use of computational resources is currently a need which can be satisfied by the three main paradigms of computing infrastructure: on-premise, cloud and hybrid. To choose one of them is often complicated, particularly for those entities that have the need of using HPC/HTC models. To many of these entities, the hybrid model could be really interesting as most of them have already some kind of on-premise infrastructure and therefore, complete migration to a cloud computing paradigm might not be the most reasonable option.

1.1 Project motivation

As expressed before, figures like 1.1 show that the dominating computing infrastructure paradigm is currently the hybrid cloud, meaning that all those entities have on-premise infrastructures and they use, in one way or another, cloud computing to complement their computational needs.

This is the particular case of the University of Cantabria which hosts several data centres to support different research needs. Some of these data centres are: 3Mares at the Faculty of Science [1], Altamira at the Institute of Physics of Cantabria [4], and Neptune/Poseidon at the Institute for Environmental Hydraulics of Cantabria (IHCantabria).

These data centres have been covering a big part of the computational needs of the different research groups and centres. In few occasions, there are needs that cannot be covered, as some lines of research occasionally require more computational power than the available at the moment, or because they need special hardware equipment the data centres do not have, like specific GPU and TPU devices.

The cost of this equipment is not negligible in terms of TCO [8]. In some cases, it might be unaffordable for small research groups or institutions, being mandatory for them to consider other options like the ones cloud computing offers.

As concluded in [10] and as figure 1.2 shows, the initial cost of cloud computing is low, as it eliminates internal workforce (e.g. system administrators) and infrastructure cost and only the time the computing resources are used is being charged. However, in the long term (third year in figure 1.2), the TCO of on-premise infrastructures becomes cheaper than using a cloud provider. Nevertheless, the optimal choice between on-premise or cloud computing depends in many other factors.

As in the case of the University of Cantabria, perhaps sticking exclusively to one of these paradigms may be too restrictive and a hybrid solution could provide an efficient and more flexible solution; making the infrastructure a little more future-proof.

The target of this thesis is to explore solutions for the mentioned hybrid environment developing a tool that can help HPC/HTC system administrators from the UC to define a hybrid cloud infrastructure. This way, researchers' jobs execution can be delegated to a cloud provider when either the local computing resources are unavailable or when a job requires special equipment that is not worth buying for whatever reason considered.

1.2 Project Organization

Due to the nature of this work being a master's thesis, it is mandatory to determine some goals that are feasible and/or achievable in the time frame provided. Therefore, the project was organized for the following steps to be taken.

1. To make a previous analysis of the state of the art and the feasibility of the project, comparing possible existing solutions to the problems this project aims to solve.

2. Using software engineering standards, develop and implementation of a framework that allows a transparent and standalone execution of a workload (job) submitted by a user using the open grid engine

workload manager. This job will be able to execute on-premise or in the cloud indistinctly, being the user the one in charge of taking the decision of where to execute it.

3. Test and evaluation of the system. A case scenario that shows the complete system working will be provided.

4. Analysis and definition of future lines of work directly related with the job done.

Chapter 2

State of the art

The concept of merging local with cloud computing has been around as early as 2008 when Jeff Barr first defined the concept "CloudBursting" [6] as the emerging pattern of moving already existing software and architecture to the cloud. In this chapter, a review of the approaches regarding this issue will be conducted. This will provide a useful insight of the tasks that need to be performed in order to help solve the issue.

There are a profuse amount of proposals for hybrid cloud/on-premise environments, which can be classified into two main currents.

The first one relies on tailored solutions dependent on the specific needs and/or architecture of one project. An example of this approach was proposed by Hao Wu et al. where they used the concept of "Cloud Federation" to describe the process of intercommunication between a local and a cloud-based infrastructure [39]. More examples of custom solutions can be found in [7] where a new experimental component, the "Cross-Cloud Federation Manager" was created as an approximation of how to manage not only hybrid cloud environments, but also how different cloud providers could interact with each other, or in [12] where a custom component was created for High Energy Physics(HEP) environments that uses the Condor job scheduler.

The alternative approach consists on either improving existing schedulers or developing a completely new one that supports interaction with cloud environments. The success of this kind of solutions highly depends on the budget available as creating (or adapting) a scheduler to support hybrid cloud environments is costly and thus, the most successful ones are the schedulers developed by companies and not the free, open-sourced ones. An example of this is the Univa Grid Engine [32] project that was forked from the original Sun Grid Engine (SGE) changing to a proprietary license and charging for the product. In contrast, Open Grid Engine, the scheduler being used and maintained at the "calderon" HPC/HTC cluster inside the 3Mares data centre at University of Cantabria, is a fork of the OpenGE project which goal is to keep it free and open source. However, despite of the great number of users, the project has been unable to maintain a relevant community of contributors and thus, it was abandoned by the end of 2012 and it is now the task of the system administrators that use it to improve it and/or maintained like in the case of the UC.

There is, however, one free and open source project that it is being widely used and currently supports some functionality regarding the cloud bursting concept even though, there is remaining work to fully support it [30]. This project is called Slurm workload manager [29], and it is being used by 60% of the top 500 supercomputers [33] though it is important to notice that, in these enterprise environments or big research groups, a support fee is usually paid to SchedMD (the company in charge of Slurm) in order to access the professional services they offer.

As stated before, Slurm provides support for CloudBursting, through a hybrid scheduling policy able to make use both the cloud and the on-premise resources. This way, if a new job is scheduled when all on-premise resources are exhausted, this job can run on the cloud.

After analysing all the options, it was decided to create a new framework for two main reasons:

1. It was a priority to maintain, for the moment, the design scheme and software infrastructure used in the HPC/HTC clusters hosted at the 3MARES data centre, without introducing new tools or systems that could have an impact over the users as they would need to adapt to the new interfaces or environments added.

2. The intention of keeping the philosophy that has characterized during many years the development and evolution of theses computers; the always open-source nature of the systems used, and the development and integration of new components by the system administrators and/or students themselves (students that have passed through here during the development of their undergraduate/graduate thesis) as a mechanism of knowledge generation within the research group.

Chapter 3

Tools used

The following chapter will describe the tools employed for the development and testing of the framework, as well as other tools that were considered useful during the duration of the project. Most of these tools are open-source and so is the code developed for this work.

3.1 Programming environment

A study conducted in 2019 by the well-known website stackoverflow.com [31] showed python as the 4th most used language between both amateur and professional developers as well as being the 2nd most loved programming language as well as being the most wanted one. In conclusion, python has been, and will probably be during the next years, among the top-five programming languages due to its relative easiness and abundance of libraries.

This last part is especially true when it comes to interaction with linux-based systems, being one of the best options for this purpose.

A growing in popularity library for python is Apache Libcloud, which exposes to the developer an API to interact with most of the existing cloud service providers in an easy and standard way. This library provides access to all the solutions these providers have, like cloud computing, object storage, load balancing or DNS services.

In this thesis, Libcloud was used to communicate with Google Compute Engine (GCE), the cloud provider chosen for this project, [16] to create, modify and destroy instances of virtual machines on demand. This interaction is based on certain policies that are defined in methods which can be modified for different needs. In section 3.2 the decision of choosing GCE is explained.

The next library used was pipenv, which unifies two well-known python tools: pip and virtualenv. The first one allows for simple package installation and management in a similar way to npm or apt (in linux environments). The second library creates a virtual environment that isolates an installation of a software piece, making it independent from the system and therefore less prominent to errors due to differences in library versions. Through this library, the user is able to install any software in different machines easily, just by typing a command.

Another important library is BlackBox, which interacts with Version Control Systems (VCS) like git and and provides safe storage for secrets (e.g. passwords or private keys) inside a repository. To do so, this library uses the GnuPG tool to encrypt the files using a keypair (public-private) scheme.

The pyyaml library was used to ease YAML parsing process(YAML Ain't Markup Language). YAML is defined in its official website as "a human friendly data serialization standard for all programming languages". YAML was chosen over other popular options like JSON because under the author's judgement it has better readability. In this project, it is used to create a configuration file that allows fast modification of the parameters that the framework needs to work correctly.

Finally, a version control system was needed during the developing phase, to keep track of the software changes and minimizing the chances of losing the source code by acting as an indirect backup. Gitlab [13] was the system chosen to host the resulting project at the GitLab of the Computer Engineering research group from University of Cantabria: https://gitlab.atc.unican.es. Due to the confidential information in this repository regarding the internal configuration of the cluster, the access to it will be restricted to the persons related to the project and to members of the jury by the use of a temporal account with username: "tribunal" and password: "kEv12KB3".

3.1.1 Bash scripting

Even though python is really easy to use to communicate with linux-based operative systems, a more efficient way of doing so is by using the native commands most of these systems have. These commands can be executed individually but most importantly, for this project, they can also be put together in a file. This file containing a set of commands is usually called a script and hence the term "Bash scripting".

According to the Free Software Foundation, Bash (Bourne Again SHell) [14] is an "sh-compatible command language interpreter" meaning that enables the use of scripting programming in a similar fashion to python. This interpreter is widely used not only in most Linux distributions but also today inside Windows 10 through the Windows Subsystem for Linux(WSL) that allows among other things to use a Unix-like command-line shell [38]. In this thesis, Bash was used to create an standalone script that will execute inside the cloud VMs in order to bootstrap them with all the necessary software and configuration.

3.2 Google Compute Engine (GCE) and Google Cloud Storage (GCS)

Among all the services that Google offers, one of them is to be a cloud provider through the Google Cloud Platform where it offers several resources like: storage, networking, big data platforms, IOT, etc.

The Infrastructure as a Service (IaaS) component inside this platform is called Google Compute Engine (GCE), which supports guest images running either Windows or Linux machines. The specifications of the hardware running these images depends on the user's choice, going from a low cost 4GB single-core configuration to the most expensive high performance configurations with multiples cores and a large amount of main memory (11,776GB of RAM) [15].

Currently, three main cloud providers dominate the market: Amazon Web Services [5], Microsoft Azure [23] and Google Compute Engine [16]

It was decided to use GCE over other competitors mainly for economic reasons because it has the most robust offer in terms of free usage, giving 300 euros in credit that can be used for 365 days (changed in August 2020 to 90 days).

To choose one cloud provider does not imply that the developed framework is bounded to it, on the contrary, it was developed to be easily adaptable to other existing or new providers.

In this thesis, GCE was used to provide the computing infrastructure to the workload manager so that jobs can be executed on the cloud. Simple instances of VMs containing a fresh installation of Debian 9 were used, deployed over GCEs zone for west Europe "europe-west2-a" to reduce latency and cost.

The second service used was the Google Cloud Storage (GCS) which, as well as the classic advantages of cloud storage (reliability accessibility, backup, etc.), offers some interesting functionalities at a reasonable price. It provides really low latency access when objects and VM accessing them are hosted in the same region and defines different storage classes (4 types) depending on the access time, making the cost proportional to it.

In this thesis, GCS was used to store packages that are not native to the debian linux environment used in the VMs created in the cloud, as well as a compressed folder containing files required for the correct configuration of this VMs. All these files are automatically downloaded from the VMs with a low latency penalty (both storage and computing resources are located in the same zone) . The use of GCS is not mandatory (though highly recommended) and it is possible to configure the framework to automatically download the necessary software from external software repositories and from local on-promise servers.

3.3 Open Grid Scheduler/Grid Engine

This open sourced scheduler first appeared in 2010 as the response from the community after the acquisition of Grid Engine by Oracle, to maintain it free and open sourced. Despite of the lack of community maintenance since the end of 2012, it is still widely used and it is, in fact, the scheduler in use by the HPC/HTC cluster "calderon" where this project has been developed. As mentioned before, keeping all the software stack that is currently in production is one of the main determinants of this project and therefore it plays and important role in the design of the framework. However, it is important to keep in mind that the framework was design to be able to support other schedulers, like Slurm, with little effort.

The architecture of openGE consists of a master node and at least one execution nodes. The master node is actively listening to new request of job executions, being in charge of managing the queues and dispatching the jobs to the corresponding execution nodes. Job and queue concepts must be introduced to explain, in a simplified way, the way openGE works.

A job represents a task to be completed on a node in the cluster while the queues are abstract representations in the shape of "containers" of the computational resources of the cluster. The scheduler policy is implemented in part through the configuration of multiple queues with different configuration, priorities, etc. [26].

The process of "running a job" starts when the users send a "characterized" job request to openGE, meaning that it includes the computational requirements that the job needs a priori (as they could vary). After this, it is openGE the one in charge of deciding, based on job information and the state of the computing nodes in that specific moment, where this job should execute and in which moment.

Once the scheduler decides where to run the job, it is sent to an execution host with a openGE client installed. After completion, the client node will inform back to the master so it can delete the job from its database and the user can access to the results of the execution. That is the simplest life-cycle of a job execution in the scheduler. However, a job can pass through many states during its life-cycle (execution, suspension, error, etc). All these transitions that a job can experience are controlled by openGE using the queues.

In this thesis, openGE will be the scheduler in charge of managing the jobs both in on-premise and cloud clients, leaving to the framework the task of providing the needed infrastructure to execute them. HIaaS-M is designed to work in harmony with openGE and to not interfere with its normal operation.

3.4 Puppet

Puppet claims itself as "The most powerful configuration management tool in the solar system" [27] Whether that is statement is true or not will remain out of the scope of this thesis but it is certainly a really useful

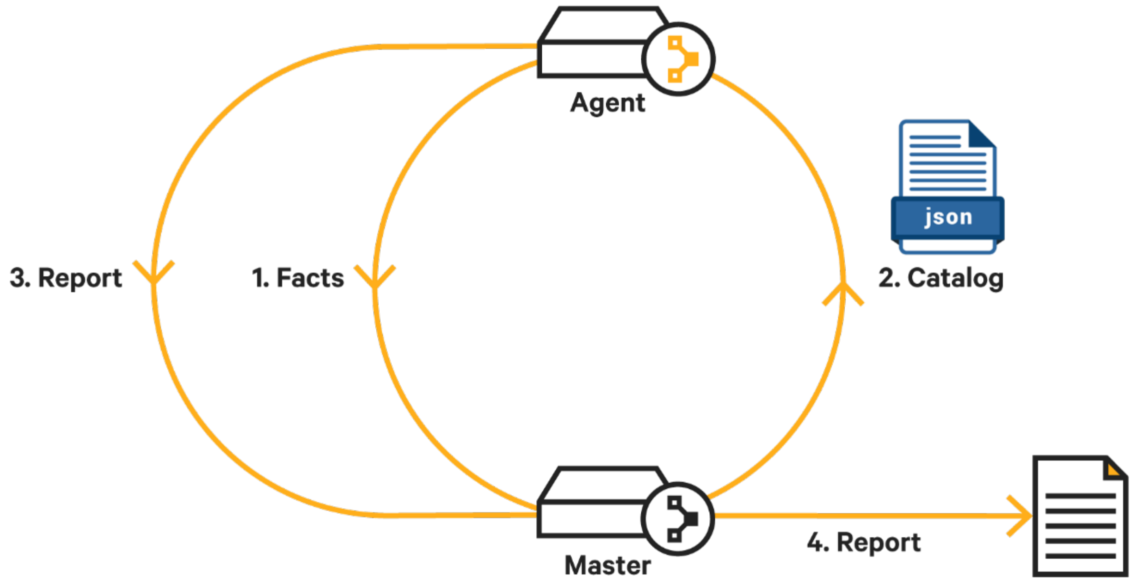


Figure 3.1: Puppet's pull mode [19]

tool in terms of easing the management of multiple hosts at once.

A typical Puppet deploy has one Puppet server and one or several Puppet clients and the way it works is (unlike other competitors like ansible or chef) by using a pull mode, meaning that the clients are the ones that send the requests to the server and not the other way around.

Puppet uses its own configuration language called DSL (Domain Specific Language) that is used to defined configuration parameters. A file containing a host configuration is called a manifest and a group of manifest creates a Puppet module.

Puppet manifests are concise and they can express variation between nodes with conditional logic, templates, and functions. Puppet resolves these on the Puppet server and gives the client a specific catalog. This catalog describes the desired stated of the client hosts in terms of configuration.

Whenever a puppet client sends a request to the server, the following steps are followed as represented in figure 3.1.

1. The Node that is running the Puppet agent (the client) collects data about itself
2. The Puppet agent sends the data to the puppet server
3. The server compiles a catalog based on data for how the node should be configured
4. The server sends this catalog back to the agent

5. The puppet agent configures the client node and reports back to the server

3.5 Single Sign On (SSO)

Single Sign-on is a centralized authentication system that allow users from one or multiple clusters to log in any of their nodes with a single ID and password (credentials), stored and managed by one or more servers. SSO is not only used in HPC/HTC environments but also in any organization that offers to its users access to services through a computer network. This system has some advantages over the traditional model of local authentication where the credentials are stored locally. Some of these advantages are listed below.

- A user can log in with the same credentials in every node inside the SSO domain
- Every node inside the SSO domain can identify and validate a user, which is useful for to node to node communications inside the same network.
- From the system administrator point of view, it simplifies the management of the users log-in information as it is stored in one or few centralized data bases.

Even though there are several ways of configuring a SSO environment, the one implemented for the HPC/HTC cluster "calderon" and the developing cluster "cluster2" inside the 3Mares data centre has the following components:

- A centralized directory that contains the identity and authorization information from every user, based on the LDAP (Lightweight Directory Access Protocol) protocol [22] and implemented by OpenLDAP [24].
- A set of administration tools provided by OpenLDAP and the web interface phpLDAPAdmin for managing the information in the aforementioned directory.
- A user authorization mechanism, in this case Kerberos, as the authentication back-end, integrated with OpenLDAP.
- Apart from this, on the client side of SSO, NSS (Name Service Switch) is used for authentication and the PAM (Pluggable Authentication Module) module as the interface used for validation between users and applications.

These elements form a client-server architecture where the server is in charge of storing the identity data of the users, offering the tools for the management of this data and providing access to the user's data, on the other hand, a client must be configured in every node that requires access to data stored inside the centralized directory which also implies that the client must be part of the SSO domain.

For the execution of this service (SSO), the HPC/HTC cluster ”calderon” has a VM exclusively designated for this task only.

3.6 OpenAFS

OpenAFS [2] is an open source distributed filesystem created at Carnegie Mellon University and widely used, especially, among academic environments. It relies on a client-server architecture and one of its key features is the centralized authentication and credential management system (implemented in many cases using Kerberos) [3] which allows access to the files stored inside these distributed systems from any machine connected to the internet.

OpenAFS, as a file distributing system, is currently being used in different clusters of the computing infrastructure (HPC/HTC research clusters) placed at the 3MARES data centre as a way to unify all the users access to their personal and projects files independently of the use (students, teachers, collaborators, etc.).

In this thesis, Puppet will be in charge of the installation and configuration of OpenAFS client in every VM created in the cloud. This configuration allows cloud machines access to local files so that the jobs executed can return their output using a file, mimicking the on-premise behaviour.

3.7 OpenVPN

A virtual Private Network (VPN) is a concept that has been around for a while and especially during these last years motivated by the increasing number of companies offering a paid service that provides anonymity to the user. Basically, a VPN acts like a point to point tunnel, creating a virtual ”physical” link between both points. This tunnel can go across the internet and allows the computers connected to its ends, to behave like there were in the same network.

One of the main uses of VPNs is the configuration of a secure link to work remotely, where the user or, in this case, student, must work from home but also needs access to the local infrastructure that the University of Cantabria provides.

This local infrastructure is secured by a firewall to prevent external and/or undesired access to it. After a secure link is opened, the user is able to reach some machines that act as ”entry points” to deeper levels. As an example, when a student connects to the external VPN of the university, they are able to connect through SSH to the ”calderon” cluster’s front-end but not to any of the developing nodes inside it. After the student opens a SSH session with one of the front-ends, they will be able to access the developing clusters like the one used for this project, ”cluster2”.

The VM created in the cloud will connect to the university external VPN the same way as a student would do. However, this SSH session is still very restricted and a newly created cloud VM won't be able to reach any of the servers inside cluster2's private network, as they either don't know the IP address of it or they don't trust it. For this reason, an additional VPN secure channel is needed.

By Hosting a OpenVPN [25] (a free and open source solution that allows the user to create its own personal VPN) inside a dedicated VM called "VPN" inside the "cluster2-master" server, a OpenVPN client (e.g. all the cloud VMs) will have their traffic routed trough the openVPN server (a trusted machine inside "cluster2"), allowing them to have a internal IP address and therefore, being able to access to private services inside "cluster2".

In this thesis OpenVPN will be used to connect from the VMs created in the cloud to the local infrastructure using internal IP addresses to communicate with several private services like OpenGE, OpenAFS or Puppet.

Chapter 4

Computing infrastructure

In this chapter, the architecture used for developing HIaaS-M will be described in detail, discussing decisions in both the hardware and software stack.

4.1 On-premise infrastructure

The Computer Engineering Group from the University of Cantabria, kindly provided the local infrastructure needed to develop this thesis, named Cluster2. This infrastructure is very similar to the “calderon” HPC/HTC cluster hosted at the 3Mares datacenter from the Faculty of Science [1]. Calderon is an active environment currently used in the University by diverse research groups such as the aforementioned Computer Engineering Group, the Real Time and Software Engineering group or the Mathematics, Statistics and Computation group.

Cluster2 is 90% similar to the ”calderon” cluster, meaning that it can be considered a small replica of it where new services, applications and frameworks are developed and tested in a controlled environment so they can be later migrated to the production environment in calderon.

Cluster2 is composed by 6 physical XEN machines divided in two groups. The first group corresponds to the ”core” of the cluster, composed by 2 of these machines called master and store. The second group is composed by 4 machines which run VMs and are the ones in charge of executing test jobs. Machines of this group receive a sequential naming in the form of v-compute1-n (one to n v-computers). The full structure of the cluster can be seen in figure 4.1.

Cluster2-master, as the name points out, is the machine in charge of the management software, such as the workload manager OpenGE, Ganglia monitoring tool or the developed framework itself. Cluster2-master is also the one in charge of running the cluster management VMs like the ”front-end” VMs where the users log-in to manage the states of running jobs, sending new jobs to execute or consulting the results of previous

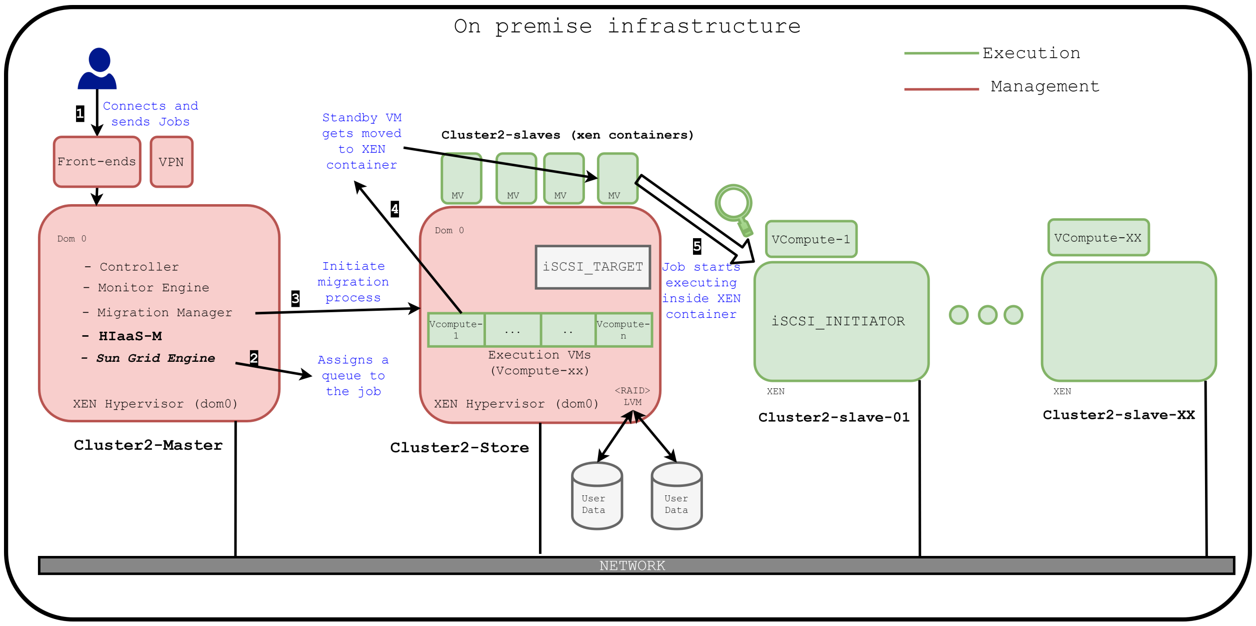


Figure 4.1: On premise infrastructure

ones, as well as the ”VPN” VM that runs the OpenVPN service which is essential for the normal functioning of the framework.

Similarly to cluster2-master, cluster2-store also executes VMs and has two roles. The first one is to be a VM-collector (similar to a pool), specifically it ”collects” all the VMs that are enabled to execute jobs (vcompute-xx). As long as these VMs do not have any job assigned (sent through the job scheduler OpenGE from the master node after a user request), they stay in a running state with the minimum virtual resources configuration (CPUs, RAM) assigned. Only when one of these VMs get assign a job to be executed is when, from a system implemented in cluster2-master, will migrate the VM (vcompute-xx) to one of the XEN containers inside the cluster2-slave-xx machines.

A second role that cluster2-store has is to store the virtual disks used by the execution VMs on-premise and to ”share” these with other XEN servers through iSCSI. The user data, as said before, is stored externally using openAFS (storage) and their user information and credentials using SSO. Both services running in VMs at the 3Mares Datacenter (including the calderon cluster) and being used as a client by the VMs of cluster2 to integrate the users and their storage so they can use the cluster as a testing infrastructure.

4.2 Cloud infrastructure

During the development of the framework, it was decided to keep the cloud architecture as simple as possible. One of the reasons behind this is the fact that every second of compute usage on the cloud is paid and

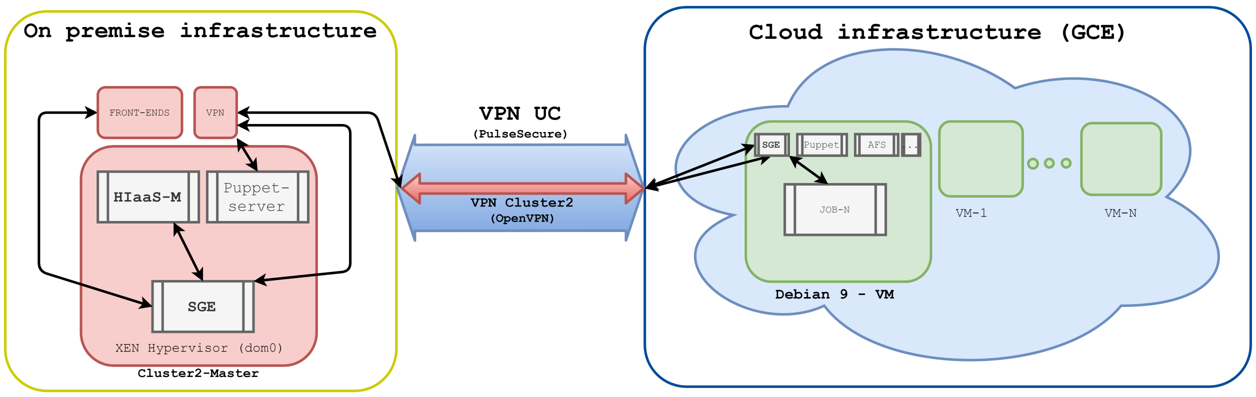


Figure 4.2: Cloud infrastructure on GCE

therefore, the time of execution had to be kept at a minimum level. This time includes the time to create the instance, the time to apply all the necessary configuration and to download all the extra software needed for the system to work as intended. The final cloud architecture can be seen in figure 4.2.

As figure 4.2 indicates, instances of virtual machines (VMs) are used to execute the jobs. This configuration might not be the appropriate one in terms of efficiency compared to alternatives like Docker containers [9], but mimics the ecosystem found in the on-premise infrastructure and therefore they are comparable, reducing the margin of errors and the time needed to apply the configuration to these new machines.

In the center of the figure two double sided arrows, one inside of the other, can be seen. These arrows represent the two different VPN connections that are needed in order to reach the private services exposed inside the cluster2 network. The first VPN connects and opens a channel with the VPN server of the UC, enabling access to the public network of the university which is protected by a firewall to prevent undesired access from the internet. The second VPN only works after the first one is successfully connected, because it opens a nested channel with the VPN machine inside cluster2, enabling the cloud VMs to have access to the private network of cluster2 and consequently, giving them access to all the private services exposed by it. Access to services like OpenGE, OpenAFS and SSO is essential as they are needed to enable job execution in the cloud.

Chapter 5

How HIaaS-M was designed

This chapter will explain how the framework was designed so it can be easily improved and expanded to new scenarios. In the previous chapter the computing infrastructure was described separately and therefore, it could be hard to visualize where the developed framework is located between all the pieces that were used during the project. That is why, before describing how the framework was developed, it might help to take a look at the figure in appendix A, where the "big picture" of the project is represented.

5.1 Design considerations

HIaaS-M was developed using the python programming language using an object-oriented approach, in order to obtain a better modularity and improving the readability of the code. Thanks to the use of Object Oriented Programming (OOP) it is easy to define classes that captures the abstract concepts the framework uses like: a VM, a workload scheduler, a manager or a parser to name some. This classes won't interact which each other but with a centralized manager class that will handle all the business logic of the framework. Thanks to this, an upcoming developer can easily change the rest of the classes/modules and adapt it to their needs. An example of this could be creating a new "Provider" class that instead of using GCE as a cloud provider uses Amazon AWS.

To make the framework flexible, all the 3 main classes implement interfaces (as shown in appendix B) exposing the methods that are necessary to make them work. For non-python users, a clarification is mandatory because, even being called an interface, python doesn't implement them in the same way as other programming languages like Java.

In the case of Java, an interface is declared by using the "interface" keyword that can be implemented later by another class through the "implements" keyword. An interface is way to provide total abstraction, meaning that the developer only specifies empty methods that must be implemented when using the interface.

```python
class Scheduler(object):
  __metaclass__ = ABCMeta
  """
  Abstract base class,
  all the derived classes must implement the following abstract methods.
  """

  def __init__(self):
      """
      There's no default set of arguments; each class should
      implement its own constructor.
      """
      pass



  @abstractmethod
  def get_sched_info(self):
      """
      Query the workload scheduler and return a list of objects
      """
      pass
```

Figure 5.1: Scheduler class as an example of an interface in Python

Another characteristic is that all the fields in an interface are public, static and final by default.

In contrast, in python these keywords do not exists but there are several ways to enforce the same or a similar behaviour. For example, to achieve an informal interface functionality, any class can be defined with empty methods that can be overridden (though it is not mandatory) when inheriting the class. The problem with this solution is that there is no way of forcing the methods to be overridden, therefore the name "informal interface".

To truly have a formal interface-like behaviour, the tools from the Python module "Abstract Base Classes" (ABCs) must be used. These tools allow the definition of abstract base classes, forcing every inheriting class to implement the methods inside or else, an exception will be raised. The exception is risen due to python's implementation of the ABC module's magic methods (methods that have two prefix and suffix) .__instancecheck__() and .__subclasscheck__(). The piece of code from 5.1 shows an example of an interface created with ABC meta class, corresponding to the Scheduler in this case. As can be seen, every class inheriting from Scheduler must implement the method "get_sched_info".

Another important aspect that had to be addressed was the delay caused by VM creation in the cloud as a consequence of being a I/O bound operation. Every time that the creation function (start_or_create_vm) is called, it waits for the VM object return value, blocking the main process progress until the deployment (creation + installation of necessary software + configuration) is ready, an action that can take several minutes. To solve this problem it was mandatory to use some form of parallelism as python's global interpreter lock does not prevent it by default. Being a I/O problem, the best way to avoid it is making use of multithreading, creating a pool of threads using the ThreadPoolExecutor module inside the "concurrent.futures" one. This executor accepts submissions of functions (and their corresponding arguments) that will be queued to be executed in a first-come-first-served basis but concurrently, each one executing in an independent thread and returning instead a future object (similar to a promise) so that the main loop can continue while the deployment of the VM is in progress.

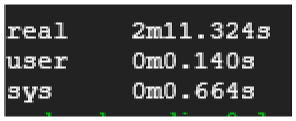
One disadvantage about this model is that the future object returned to main loop does not guarantee that the VM creation process is successful. For this reason, the main loop must verify during each iteration the state of the VMs, checking if the machine was created correctly. If any error is found, the required actions are taken, deleting the VM from the list of managed VMs if necessary.

Finally, one more design consideration must be mentioned, which is the decision of using a start-up script as a way of bootstrapping the cloud VMs with all the necessary software and configuration instead of using other methods, like creating a custom image with all the necessary software.

Many cloud providers allow to create custom images that can be used when creating a new instance. For example, GCE allows the creation of a new instance in which all the necessary software and files are downloaded manually as well as all the desired configuration to finally create a customized image based on the boot disk attached to that instance, creating a snapshot using the state VM was left in. However, this methodology would imply a certain dependency on the cloud provider because, even though there are ways to migrate these images between alternative providers (for example, AWS), it cannot be assumed that all the cloud providers are going to implement the same migration policy. Through the proposed bash-script bootstrapping, future compatibility issues are avoid and the framework independence from the cloud provider i maintained.

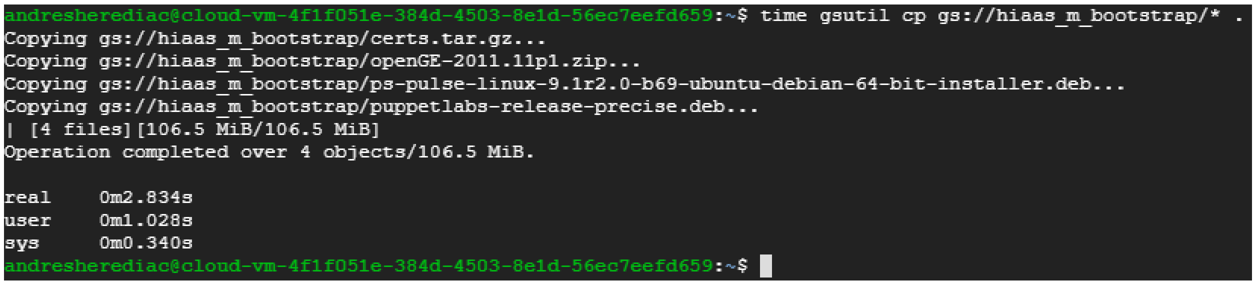
This bash script (start-up) is an independent file which is sent automatically over a SSH connection after the creation of the VM in the cloud. It installs the necessary software by executing commands in the machine. Currently, the script is only dependent on the operative system, in this case debian distributions (debian, mx linux, deepin, ubuntu, etc) which most if not all of the cloud providers offer as an option when creating a VM. It can also be easily modified to add or delete additional software or configuration (thought this part belongs more to puppet, it can be done by using the script) without having to create a whole new image.

During the development of the framework, the utilization of this start-up script revealed some limitations,



```
real    2m11.324s
user    0m0.140s
sys     0m0.664s
```

Figure 5.2: Download time of the OpenGE configuration folder under two VPNs



```
andresherediac@cloud-vm-4f1f051e-384d-4503-8e1d-56ec7eefd659:~$ time gsutil cp gs://hiaas_m_bootstrap/* .
Copying gs://hiaas_m_bootstrap/certs.tar.gz...
Copying gs://hiaas_m_bootstrap/openGE-2011.11p1.zip...
Copying gs://hiaas_m_bootstrap/ps-pulse-linux-9.1r2.0-b69-ubuntu-debian-64-bit-installer.deb...
Copying gs://hiaas_m_bootstrap/puppetlabs-release-precise.deb...
| [4 files][106.5 MiB/106.5 MiB]
Operation completed over 4 objects/106.5 MiB.

real    0m2.834s
user    0m1.028s
sys     0m0.340s
andresherediac@cloud-vm-4f1f051e-384d-4503-8e1d-56ec7eefd659:~$
```

Figure 5.3: Download time of the OpenGE configuration folder using a GCS bucket

but all of them can be prevented or minimized.

- In the first place, the script depends on the public debian repositories as well as on external servers to retrieve the software like pulse, openvpn, etc. If for example, the software version that must be downloaded ceases to exist, the creating process will crash the whole system because the framework won't validate the VM and will be blocked in an infinite loop of killing and creating new VMs. This problem can be easily addressed replicating the required software in a server with high availability like a content delivery network (CDN) server [34] or even hosting it inside a public ftp server at the same data centre where the framework is used.

- Second, the script is slow because it has to download several pieces of software. A particularly slow process corresponds to the the OpenGE client and configuration from the university servers, as it does it behind two VPNs connections, which highly impacts the bandwidth and thus the downloading speed. This problem is specific to our environment, and can be solved using the same method mentioned above if the hosting server offers higher downloading speeds.

- Finally, the start-up script allows for installation and configuration processes only during the creation of the VM. Even though it could be possible to create cron routines to periodically perform certain actions, it is really limited when a configuration change is needed a posteriori. This issue was solved using Puppet as the configuration manager.

As mentioned before, two of the limitations of script-based VM creation can be avoided making use of some kind of external storage server and thus, an alternative version of the start-up script was created to work WITH GCS. Using a bucket located in the same zone as our created VMs, an important improvement in the deployments speed was achieved. 5.2 shows the time command output that downloading only the OpenGE configuration folder form the university generates, and 5.2 shows the time taken in downloading the same folder plus two packages and another configuration folder from the GCS bucket thus reducing the time in more than 128 seconds by using this alternative, which means it is 46,34 times faster

Unfortunately, this speed improvement generates a dependency on GCS because the main reason behind it is that the bucket and the VMs share the same region and it can be easily access from inside the VMs (there is no need for authentication, the download is performed from within the VM which is already logged inside GCE). This could be considered a disadvantage or even a problem and that is why the use of GCS is only recommended and not mandatory.

Another downside could be the price, because maintaining objects in cloud can be expensive and unlike the VM instances, storage cost monthly money whether is being used or not. However, the billing example that Google gives in the official GCS pricing website [17] seen in figure 5.4, shows that for less than two extra dollars per month, the framework would be able to download the files needed to deploy 50k VMs. This extra cost could be really worth it, especially considering the speed-up obtained.

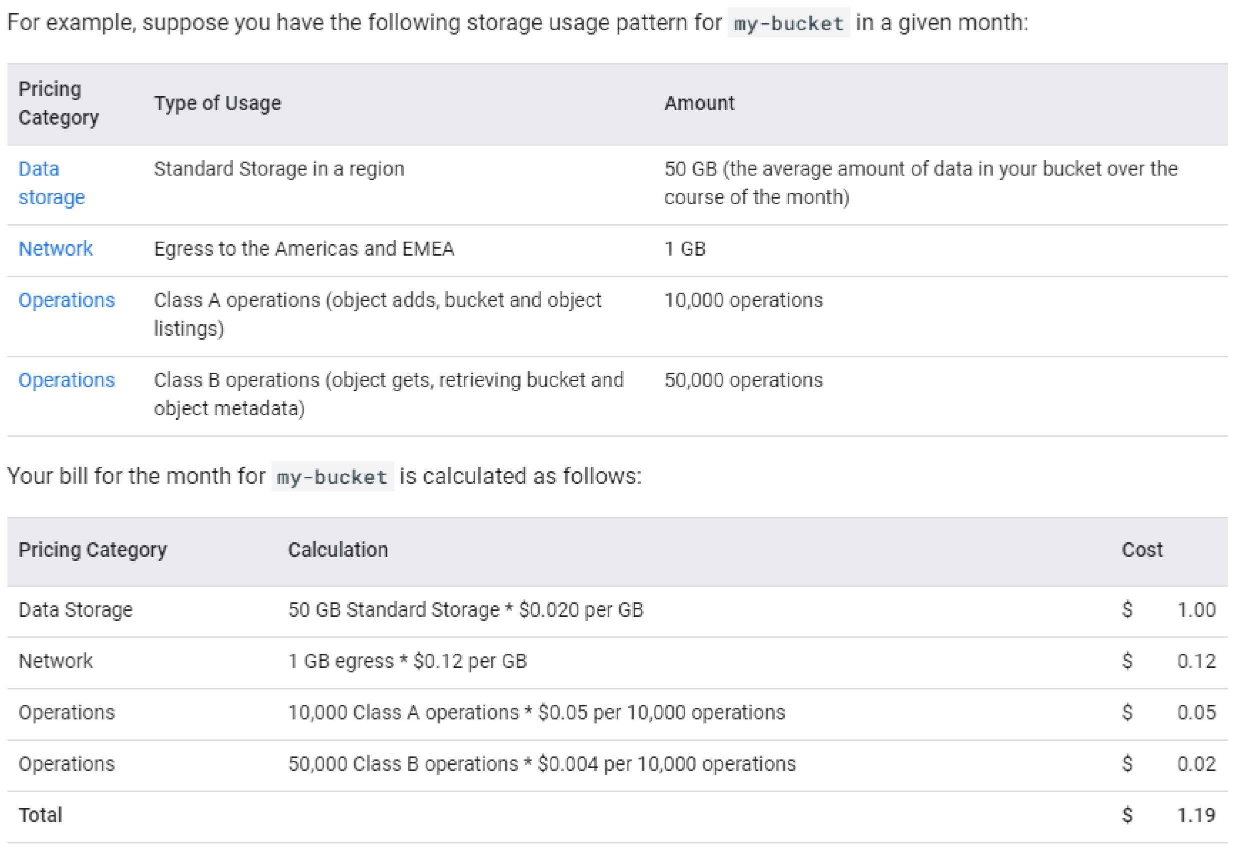
5.2 Folder structure

To best organize the project and following software development recommendations, the project is divided in a series of folders and files as shown in figure 5.5. This structure is very simple but enough to keep the different responsibilities isolated and well identified.

Starting from the parent directory, the package folder cloud_provider contains the implementation of the interface for two providers: dummy and GCE, both belonging to the Libcloud library. The dummy provider is a mock-up provider that returns instances of fake cloud nodes which were used to test other functionalities in the manager without having to spend real money. The GCE file implements the real Libcloud interface used to talk with GCE and to create real nodes in the cloud. This interface provides three "basic" functions: start_vm, stop_vm and update_vm. Including an additional interface to any new cloud provider requires the implementation of at least these functions. This class, the same as the classes provided, has to follow an OOP paradigm.

Continuing with the packages folders, scheduler defines and contains the interface that every scheduler or workload manager must implement in order to interact with the system. In this case only one method is mandatory, called get_sched_info. This method should return the information about the state of the jobs (workload) in the system. For this project, an interface that interacts with OpenGE was implemented by parsing the xml output of the qstat command, which returns the state of the queues, and returning a list of them that can be retrieved from the manager.

Next are the folders secret and util. The first one contains the encrypted private key used to authenticate


For example, suppose you have the following storage usage pattern for `my-bucket` in a given month:

Pricing Category	Type of Usage	Amount
Data storage	Standard Storage in a region	50 GB (the average amount of data in your bucket over the course of the month)
Network	Egress to the Americas and EMEA	1 GB
Operations	Class A operations (object adds, bucket and object listings)	10,000 operations
Operations	Class B operations (object gets, retrieving bucket and object metadata)	50,000 operations

Your bill for the month for `my-bucket` is calculated as follows:

Pricing Category	Calculation	Cost
Data Storage	50 GB Standard Storage * $0.020 per GB	$ 1.00
Network	1 GB egress * $0.12 per GB	$ 0.12
Operations	10,000 Class A operations * $0.05 per 10,000 operations	$ 0.05
Operations	50,000 Class B operations * $0.004 per 10,000 operations	$ 0.02
Total		**$ 1.19**

Figure 5.4: An example of the monthly cost of a 50 GB bucket in GCS

```
HIaaS-M
├── cloud_provider
│   ├── dummy.py
│   ├── GCE.py
│   └── __init__.py
├── config
│   └── config.yaml
├── scheduler
│   ├── gridengine.py
│   └── __init__.py
├── secret
│   └── hiaas-m-570dfe44cfb5.json.gpg
├── util
│   ├── config.yaml
│   ├── startup_script_gcs.sh
│   ├── startup_script.sh
│   └── utils.py
├── __init__.py
├── __main__.py
├── manager.py
├── Pipfile
├── Pipfile.lock
└── README.md
```

Figure 5.5: Folder structure of the project

with GCE and it is the place where all the vulnerable information should be stored after encryption. The folder util contains the bash script executed for bootstrapping all the new VMs and a series of helping classes like JobInfo and VmInfo which are used to encapsulate these abstract concepts so they can be reusable for newer additions (e.g. schedulers or providers)

Finally, the manager script contains the main logic of the framework inside the equally named class. This class declares an interface which has to be implemented in order to define the policies that will decide 3 things: if a job should be executed in the cloud, when are new VMs needed and finally whether a VM can be stopped or not. Apart from this interface, it also implements several methods that are invoked inside a unique loop main method where it will interact with all the other components in order to create and destroy new VMs according to the implemented policies.

5.3 Configuration

This framework would not be very useful if there would not be a way of configuring the basic parameters that are needed to migrate it to other environments and thus, a YAML configuration file was created. This YAML file allows the system/cluster administrator to configure the environment in a quick and very understandable way.

The multiple parameters that must be configured for the framework to work properly are explained below

- logLevel: used to configure the logging level used by the framework, based on the logging python module with available levels: CRITICAL, ERROR, WARNING, INFO, DEBUG and NOTSET. For a production environment, it is recommended to use INFO or superior level, instead of DEBUG.
- MANAGER->Max_vms: controls the maximum amount of VMs that can be created by the manager. It is really useful to keep the cost under control in case an unexpected glitch or bug creates an infinite loop of creating new VMs.
- MANAGER->Loop_delay: controls the time of the manager's cycle, meaning that every 'X' amount of time the loop has to start again (the time starts counting once the previous cycle ends so it is not an exact measure). It is expressed in seconds.
- SCRIPT->Name: name of the script that has to be sent to the cloud created VMs for deploying. The framework will look for it inside the 'util' folder so it is important to remember to place it there.
- GCE->KEYS->ssh_username: username that will be used to log in and execute the commands after the VM is created. It is mandatory for this user to have enough privileges to execute commands as a superuser (sudo)
- GCE->KEYS->auth_account: account that is used for the GCE module to auth with the cloud and to e able to create the VMs; this account has project level access and should have enough rights to create/destroy/modify VMs inside GCE.
- GCE->KEYS->auth_key: private key associated to the account mentioned above and to the project. It has to be located inside the 'secret' folder so it can be encrypted and passed through git using blackbox.
- GCE->KEYS->ssh_public_key_path: path to the machine's public key that will be used to connect without the need of using a password to the VM created in the cloud. In this case a path is used because most of the times this key is already existing in most of the linux based servers.
- GCE->KEYS->ssh_private_key_path: path to the machine's private key complementing the public key used in the previous parameter.
- GCE->Project: name of the project inside the GCE management console.
- GCE->Region: general region where the project will create the VMs. It is recommended to use physically close location in order to reduce as possible the latency of access to the servers.

- GCE->Resources->Size: selection of size for the VMs created in GCE. The full size list can be checked at the official GCE website [16].
- GCE->Resources->Image: the image flavour that will be used as a boot disk. In this case, as explained in previous sections, there is a restriction of using only debian based distributions.
- GCE->Resources->Location: exact location where the VM will be create. Usually all the regions have more than one location, for example London (europe-west2) that has 3: a, b and c.
- GCE->Resources->Preemtible: boolean value that configures whether the VM should be preemtible (will not long longer than 24 hour and can be turned off automatically by Google if there is a need of its computing resources). This parameter totally depends on the type of jobs that are being executed or if, as in this thesis, tests are being made and therefore it can save money and time as they turn off automatically).

Figure 5.6 shows the configuration file used for this thesis as a representative example for the parameters explained before.

```
appName: HIaaS-M
logLevel: DEBUG

MANAGER:
  Max_vms: 8
  Loop_delay: 60

SCRIPT:
  Name: startup_script.sh

GCE:
  KEYS:
    ssh_username: andresherediac
    auth_account: manager@hiaas-m.iam.gserviceaccount.com
    auth_key: hiaas-m-570dfe44cfb5.json
    ssh_public_key_path: ~/.ssh/HIaaS.pub
    ssh_private_key_path: ~/.ssh/HIaaS
  Project: hiaas-m
  Region: europe-west2
  Resources:
    Size: n1-standard-1
    Image: debian-9-stretch-v20190916
    Location: europe-west2-a
    Preemtible: True
```

Figure 5.6: Configurable parameters

Chapter 6

How HIaaS-M works

This chapter delves into the functioning of the software developed, explaining how it works internally, how it communicates with the previously existing architecture to manage the cloud VM instances, how it keeps the consistency among all the VMs and how it deploys all the necessary base software in them.

6.1 Main loop

As explained in the previous chapter, the main class is the one called manager. This class centralizes the logic of the framework, being in charge of communicating with the rest of the classes and checking their correct behaviour. The manager is independent of both the cloud provider and the workload manager.

One of the best ways explain the operation of this class is by using a flow chart. The flowchart of the manager class can be seen in 6.1 and as it shows, after the initialization of parameters/attributes (either coded or from the configuration file) it performs a check on the workload status, meaning that is going to retrieve from the workload manager the status of the queues and running jobs. It returns a list of all the jobs and their status, checking if they need to be executed in the cloud, or if there is any job that finished and does not need to be managed any further and ultimately checking for job status changes.

Right after this, it collects information from the cloud provider interface to update the status of the VMs that is managing (check_vm_status). This is done to ensure the local VM list is synchronized with the one in the cloud and if that is not the case to update it.

After these checks are done, and all the lists with jobs and VMs are synchronized with one another, the manager checks if new VMs are required. To do this, it is going to check the overridden method check_vm_policy. If certain conditions are met, the manager will start the process of creating a new VM in GCE, continuing in parallel with the execution of main loop.

The next step is to check if there is any VM in the cloud that can be stopped and if that is the case, forcing

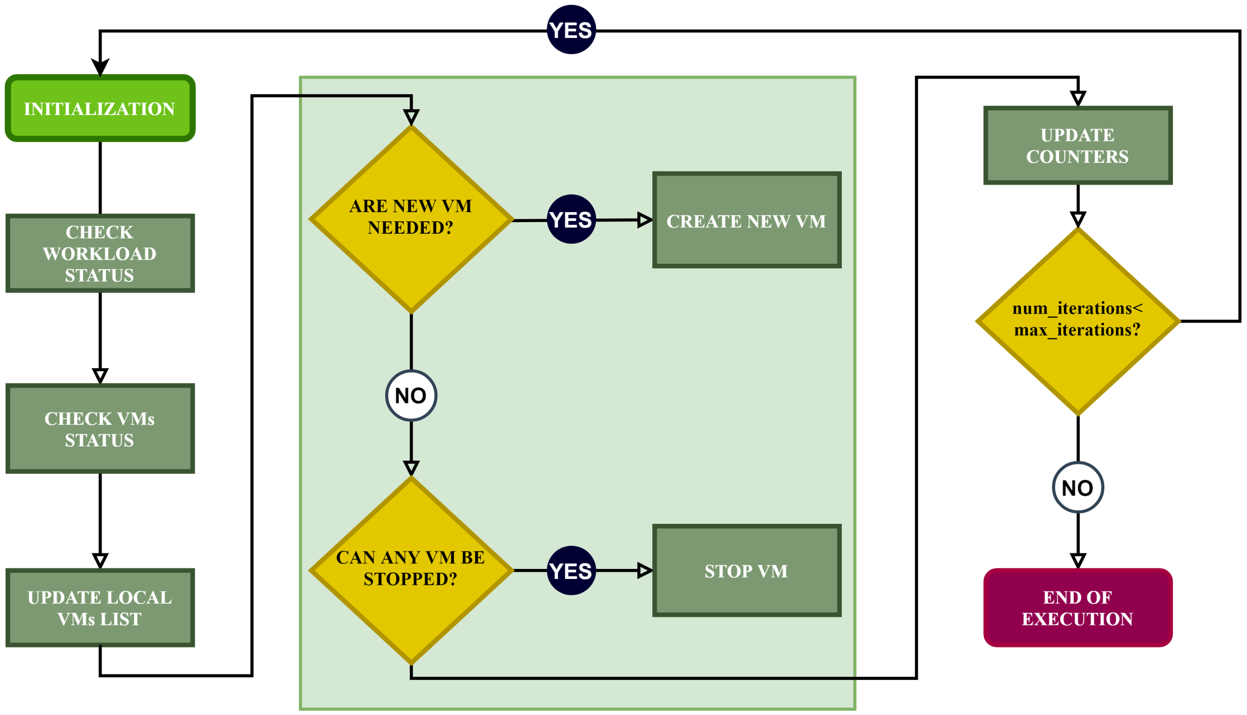


Figure 6.1: Main loop flow chart

the cloud provider to destroy it. This step is really tricky and it should be used with caution. For this thesis, it was considered that a VM can be destroyed when there is no job assigned to it after waiting a programmed delay to ensure it is not destroying a starting VM. This step is necessary and really important as it acts similarly to a garbage collector, stopping and deleting all the VMs that could have remained running but without any active job in them or that are in an undesired state. For example, if GCE decides to kill a VM for any reason, the framework won't get notified and without these step, it would think that is still running. The importance of this step/method gets increased considering that if there are active VMs remaining in the cloud, they will incur an added and unnecessary cost in the monthly bill. Given the importance of this method, as well as all the policies implemented, it is susceptible of further improvement, remaining open as a future line of work.

Finally, the main loop is going to update the counters used to track the execution (like the number of iterations) so it can finally check whether it should continue running or to just end the execution there. This loop will execute either the number of iterations configured or infinite times if this value is 0.

6.2 Scheduling jobs in the cloud

Now that the big picture of the framework is described, the next sections will focus in explaining with a higher level of detail the operation of the 4 key components or interfaces of the framework:

1. The workload scheduler interface
2. The cloud provider interface
3. The start-up script
4. The puppet implementation

It must be noticed that, even though the framework was created to be able to use in different scenarios, the explanation that follows is based on the specific implementation and configuration made for the clusters of the 3Mares data centre and thus, it can suffer variations if other components are used. As a reminder, for this thesis GCE and OpenGE were used.

6.2.1 The workload scheduler interface

As seen in the main loop, one of the first steps is to check the status of the workloads in the queue system, in this case OpenGE.

In the section explaining OpenGE, it was mentioned that the queues are well configured containers with dozens of parameters that the scheduler uses to locate jobs in a computational environment. Once the scheduler assigns a job to a queue, this job can be executed in one of the nodes (linked to the queue), depending on certain configuration parameters and/or the load of the system in that precise moment. Over the life-cycle of a job, this can pass through several states like pending, executing, suspended, error, deleted, etc...

To retrieve information about the state of the system, OpenGE provides a series of commands, as well as graphical interface (that won't be used in this case). One of these commands is qstat; from the qstat manual page "The qstat command is used to request the status of jobs, queues, or a batch server".

This command will be used in combination with two arguments that were crucial for the implementation of the interface that interacts with the scheduler: '-xml' and '-r'. The argument '-r' shows the required resources the job specified, like the queue where it would like to execute (OpenGE will still decide at the end), and the '-xml' argument shows the output of the command in the xml format. An example of the output produced by this command with those arguments can be seen in figure 6.2.

For the implementation of the custom parser, needed to check the state of the cluster, a github repository called qstat created by the user 'relleums' was forked [28]. The code from this repository could not be used

```
<job_info xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <queue_info>
 </queue_info>
 <job_info>
        <job_list state="pending">
         <JB_job_number>156895</JB_job_number>
         <JAT_prio>0.50500</JAT_prio>
         <JB_name>STRESS</JB_name>
         <JB_owner>andres</JB_owner>
         <state>r</state>
         <JB_submission_time>2020-11-23T17:20:34</JB_submission_time>
         <queue_name></queue_name>
         <master></master>
         <slots >1</slots>
         <full_job_name>Test.cloud</full_job_name>
         <hard_req_queue>cloud.q</hard_req_queue>
        </job_list>
 </job_info>
</job_info>
```

Figure 6.2: Example of qstat -xml output

directly and required some modifications. It contained a bug that makes the framework crash as it does not consider a single job waiting to be executed and it did not support the '-r' argument.

The fork version of the repository was modified and uploaded to the gitlab server where the framework is stored and the 'Pipfile' file that pipenv uses was changed accordingly to point at this repository.

Parsing and interpreting the qstat command output correctly, the manager is now able to ask the scheduler implementation (OpenGE) for a list of the jobs inside and outside the queues and to check if in a given moment there is any job matching with the implemented policy, meaning that this job has to be executed in the cloud.

This policy can have many implementations or interpretations and it will be mostly divided in two groups: static and dynamic policies, being a static policy the one that has rather simple conditions and a deterministic output in contrast to a dynamic policy that stablish a series of rules that have different outcome depending on the state of the workload manager and other factors (it could even be random). This second kind of policies can turn really complicated and the development of an efficient one could lead to a future line of work for this project.

For this thesis, with the intention of keeping it simple as it is out of the scope of the project, we decided to use a static policy. In this case a special queue was created in the local OpenGE server called cloud.q,

```
# When are new VM needed:
# in this case when we have more candidates than VMs created
  def check_vm_policy(self):
      if len(self.candidates) > len(self.vms):
          return True
```

Figure 6.3: Static cloud policy used by the framework

```
ComputeEngine = get_driver(Provider.GCE)
# Creation of the driver for the GCE provider, GCE
self.provider = ComputeEngine(auth_account,  str(full_path), project=project,
    datacenter=datacenter)
```

Figure 6.4: Example of usage of the cloud provider interface

meaning that every job which destination is this queue should be executed in a cloud VM. This static policy implementation used for this project can be seen in figure 6.3.

6.2.2 The cloud provider interface

Whenever there is a match with the ”cloud policy”, the manager will start a new VM in GCE using the cloud provider implementation.

The process of launching a new VM instance in a cloud provider, GCE in this case, implies taking several steps to ensure it has the right environment to execute the job and that it has the ability of full communicating with the on-premise servers at the development clusters (cluster2).

The first step consists on using Libcloud to communicate with GCE using an account with enough permissions (configured in the YAML configuration file). In this case the manager account will be used for simplicity.

Libcloud logs into GCE using the credentials provided, in this case the email and the json file generated by GCE that contains the key associated to the account. This login process uses the OAuth2 framework [20] to ensure it is secure. Thanks to Libcloud, the code needed to connect to GCE can be as simple as the one showed in figure 6.4.

After a successful login, the next step is to enable the remote connection with the machine executing the framework (HIaaS-M), Cluster2-Master in this case. To do so, the installation of the public key belonging to Cluster2-Master is needed.

The way Libcloud deals with this procedure is by sending additional metadata inside the request done to create a VM in a form of a python dictionary. Inside this metadata the public key as well as the ssh

```
# Multiple-step that installs our SSH key and executes the deployment
# script that installs the base software
metadata = {
    'items': [
        {
            'key': 'ssh-keys',
            'value': '{}:␣{}'.format(ssh_username, PUBLIC_SSH_KEY_CONTENT)
         }
    ]
}
```

Figure 6.5: Metadata sent to the cloud VMs

username will be sent to the VM. For clarification, an example of this can be seen in figure 6.5.

After going through these steps, the manager has a working VM instance on GCE that can keep in its local list of managed VMs. However, this VM is a fresh install of Debian 9 and therefore does not have neither any of the required software that grants the execution of a job nor the ability to communicate with the university's machines. Is in this case where the start-up script comes in handy.

6.3 The start-up script

The goal of the start-up script is to prepare the needed software, configuration and VPN connections so that the cloud virtual nodes can be at the same network level as the ones of the on-premise infrastructure. Being at the same level and having the right software, the new cloud nodes can be detected as execution nodes by the on-premise scheduler OpenGE.

Once this instance is created and launched by the cloud provider, this deployment script written in bash starts the following sequence in order to prepare it to run an incoming job:

- Updates the packets from the linux distribution repositories
- Installs the following software: sshpass, resolvconf, Pulse Secure vpn,OpenVPN and a Puppet client
- Configures the puppet client with information about the Puppet server installed in Cluster2-Master so it is able to connect to it
- Connects to the exterior university VPN service (VPN UC) entering the university's public network which allows to reach the Cluster2-Master and the OpenVPN server installed in it.
- Adds Cluster2-Master as a known host and connects to it to get a copy of the certificates needed to establish a connection with the OpenVPN server

- Creates the required environment in order to use OpenVPN (users, groups, configuration, etc.)
- Connects to the OpenVPN server (VPN Cluster2) to obtain an internal IP address which is essential to enable communication within the cluster2 infrastructure.
- Initializes the Puppet client service, leaving the control of the rest of the configuration to it.
- Finally, a clean-up is done, refreshing the DNS entries that could have been compromised during the process of connection to the two VPNs ('VPN UC' and 'VPN Cluster2') and removing the initialization script itself.

6.4 The Puppet's role

As seen in the infrastructure chapter, a Puppet server instance is installed inside the Cluster2-Master server.

This Puppet server is the one in charge of managing the VMs full set-up, because it does not only allows to apply configuration in terms of, for example, parameters or configuration files, but also allows to install and configure many pieces of software like: nginx, apache, java, elastic search, openAFS, OpenGE client, SSO client, etc., that could be needed to set the right environment to execute a job. For the project's particular case, Puppet is employed to install the required software to access the local files distributed by OpenAFS.

Delegating this responsibility to Puppet, the administrator can easily fulfil the requirements of nearly any job, providing an overall solution with great flexibility and adaptability to new environments.

As stated in the tools chapter, Puppet uses manifests and modules to stablish host configuration. In this particular project, a simple manifest called worker.pp was used containing the code showed in figure 6.6.

This manifest invokes the afs module developed by kodguru [21] and pushes it into the clients attached to the Puppet server. This module requires some customization in order to properly configure openafs according to the project's requirements, concerning yaml file naming and location (PATH). The configuration folder contains configuration files for different operating systems, but debian is not included. It was necessary to adapt the configuration file for the Ubuntu distribution, renaming it as Debian.yaml. This file will contain the configuration needed to configure afs in the puppet clients.

Figure 6.7 shows all the parameters used to configure this module, including all the packages that must be installed. Among the several parameters that the module accepts, the most relevant ones are:

- afs::cell - describes the address of the cell in the server.
- afs:afs cellserverdb - list of database server machines in the local cell.

```
node default {

include afs

    class { '::nfs':
        server_enabled => false,
        client_enabled => true,
        nfs_v4_client => true,
        nfs_v4_idmap_domain => $::domain,
    }
    nfs::client::mount { '/home':
        server => '192.168.0.100',
        share => 'home',
    }
}
```

Figure 6.6: Manifest used to configure the client hosts

```
---
afs::init_script: /etc/init.d/openafs-client,
afs::init template: openafs-client-Ubuntu-init

afs::afs_config_path: /etc/openafs
afs::cache_path: /var/cache/openafs
afs::config_client_dkms: true
afs::config_client_path: /etc/default/openafs-client
afs::package name:
    - openafs-client
    - openafs-doc
    - openafs-modules-source
    - openafs-modules-dkms
    - openafs-krb5
    - dkms
afs::cell: atc.unican.es
afs::afs_cellserverdb: |
    >atc.unican.es
    192.168.0.105       # is.local
---
```

Figure 6.7: OpenAFS Puppet module configuration

This manifest also uses the module nfs from the official puppet repositories. In the code 6.6, the second part shows how a nfs client is configured through the nfs class and its corresponding parameters. This nfs configuration is required by the cloud VM to mount a remote network folder that will provide the files needed to configure OpenAFS.

This manifest example shows how two essential pieces were configured: OpenAFS and NFS, but its valid for any other piece that the system needs, like the openGE client, the SSO client, etc. In many cases, already developed modules can be found. If not, developing and integrating a new module into a Puppet server will be necessary.

In summary, this chapter introduces the main operation of the developed framework, HIaaS-M, as well as a step by step description of the different elements that "collaborate" to set up an operative cloud VM. After these steps, the following elements are configured and ready to work:

- The workload scheduler interface.
- The cloud provider interface.
- The start-up script.
- The puppet implementation.

At this point, a newly created cloud VM is ready to remotely execute a job sent from the on-premise openGE master inside the cluster2-master server. Without these steps, a job sent to the queue 'cloud.q' would stay forever inside openGE in a 'qw' state, waiting to have the needed computational resources to be executed.

Chapter 7

Proof of concept and results

In this chapter, a use case in the execution of the HIaaS-M framework will be performed in order to demonstrate how it works and what to expect from it.

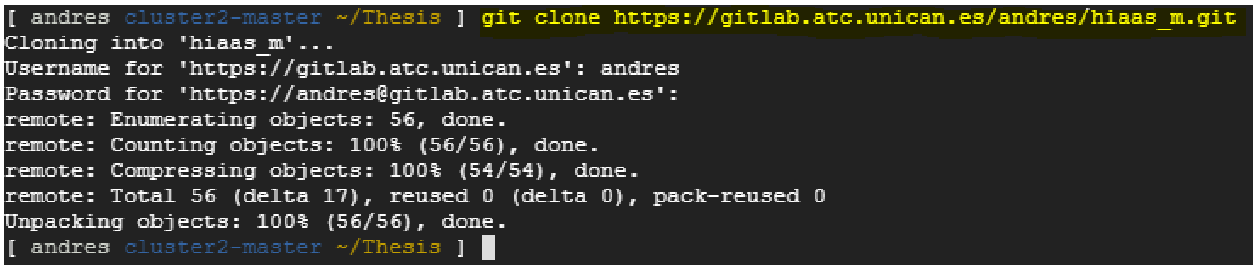
The software layer containing the framework for this test was placed inside the development cluster (cluster2), where it was developed, specifically in the cluster2-master node which, as said before, it is the brain of the system and contains (apart from the framework) all the necessary third party software and configuration that allows it to communicate with other nodes and to manage the execution of jobs in the on-premise infrastructure.

The framework will periodically query (using the Loop_delay parameter set in the configuration file) the OpenGE master's instance, checking the job lists and queues status, checking for matches with the implemented policy to decide whether a job should be executed on the cloud and if so, providing the necessary infrastructure, making OpenGE aware of the new computing resources available to execute jobs. OpenGE is still in charge of the workload management, providing the framework only the required VMs for a job execution.

7.1 Requirements

Before beginning with the installation, it is important to notice that HIaaS-M was tested with the following versions of the required software:

- Debian 9-4.9.189-3+deb9u2 (2019-11-11)
- Puppet server 6.9.0+
- OpenAFS 1.6.20+



```
[ andres cluster2-master ~/Thesis ] git clone https://gitlab.atc.unican.es/andres/hiaas_m.git
Cloning into 'hiaas_m'...
Username for 'https://gitlab.atc.unican.es': andres
Password for 'https://andres@gitlab.atc.unican.es':
remote: Enumerating objects: 56, done.
remote: Counting objects: 100% (56/56), done.
remote: Compressing objects: 100% (54/54), done.
remote: Total 56 (delta 17), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (56/56), done.
[ andres cluster2-master ~/Thesis ]
```

Figure 7.1: Cloning the HIaaS-M repository

- OpenVPN 2.4.8+
- OpenGE2011.11p1
- Python 3.5+
- pipenv 2020.6.2+

7.2 HIaaS-M initialization

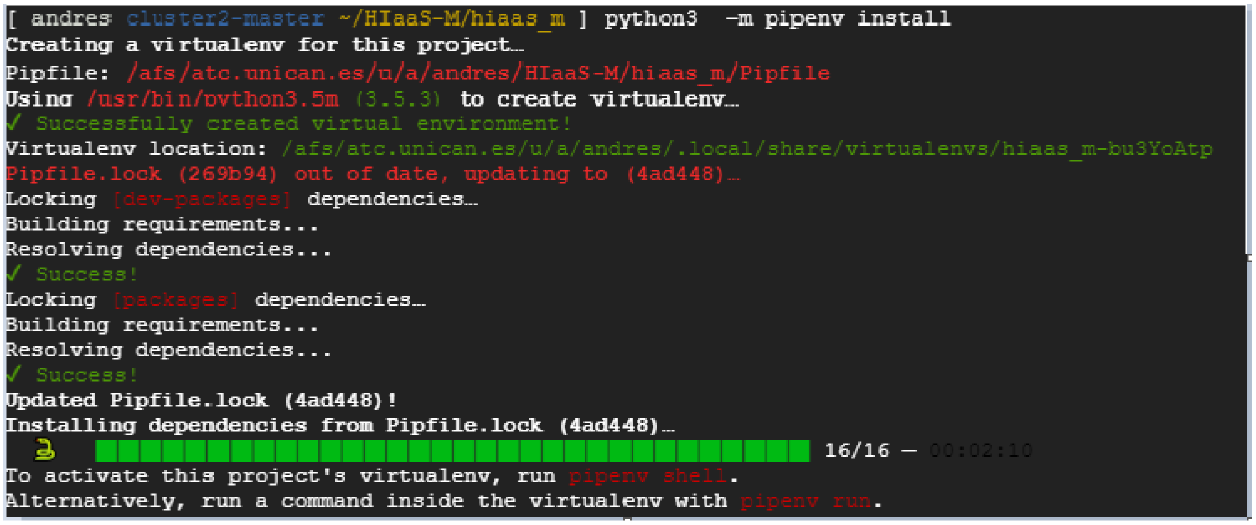
After all these checks up, the next step consists on cloning the repository from the Gitlab platform to a local folder, as seen in figure 7.1. To do this, the command git clone should be used.

After downloading the source code, the next step consists on the decryption of the private key used to connect to our cloud provider or, like in this case, to explicitly provide that key. This key must be located in the "secrets" folder.

Then, a virtual environment is created using the command "python3 -m pipenv install" from the parent folder as figure 7.2 shows. Thanks to the library pipenv, all the necessary libraries are automatically installed inside the created environment.

The final installation step requires the modification of the configuration file according to the environment requirements. In this particular case the configuration previously showed in figure 5.6 will be used.

Having the right configuration, the right keys and all the dependencies installed, the manager can get started. To do so, the virtual environment is activated using the command "pipenv shell" (beware of executing this inside the project's folder). Then, from the parent folder of the project, the command "python3 -m hiaas_m" starts the manager's execution. Changing to the parent folder is necessary because python will try to find a hiaas_m module inside the current folder (which does not exist). Naming the folders with the same name as the modules that they contain is the default behaviour of python and it is not recommended to change it.



```
[ andres cluster2-master ~/HIaaS-M/hiaas_m ] python3  -m pipenv install
Creating a virtualenv for this project…
Pipfile: /afs/atc.unican.es/u/a/andres/HIaaS-M/hiaas_m/Pipfile
Using /usr/bin/python3.5m (3.5.3) to create virtualenv…
✓ Successfully created virtual environment!
Virtualenv location: /afs/atc.unican.es/u/a/andres/.local/share/virtualenvs/hiaas_m-bu3YoAtp
Pipfile.lock (269b94) out of date, updating to (4ad448)…
Locking [dev-packages] dependencies…
Building requirements...
Resolving dependencies...
✓ Success!
Locking [packages] dependencies…
Building requirements...
Resolving dependencies...
✓ Success!
Updated Pipfile.lock (4ad448)!
Installing dependencies from Pipfile.lock (4ad448)…
                                                          16/16 — 00:02:10
To activate this project's virtualenv, run pipenv shell.
Alternatively, run a command inside the virtualenv with pipenv run.
```

Figure 7.2: Installation of HIaaS-M

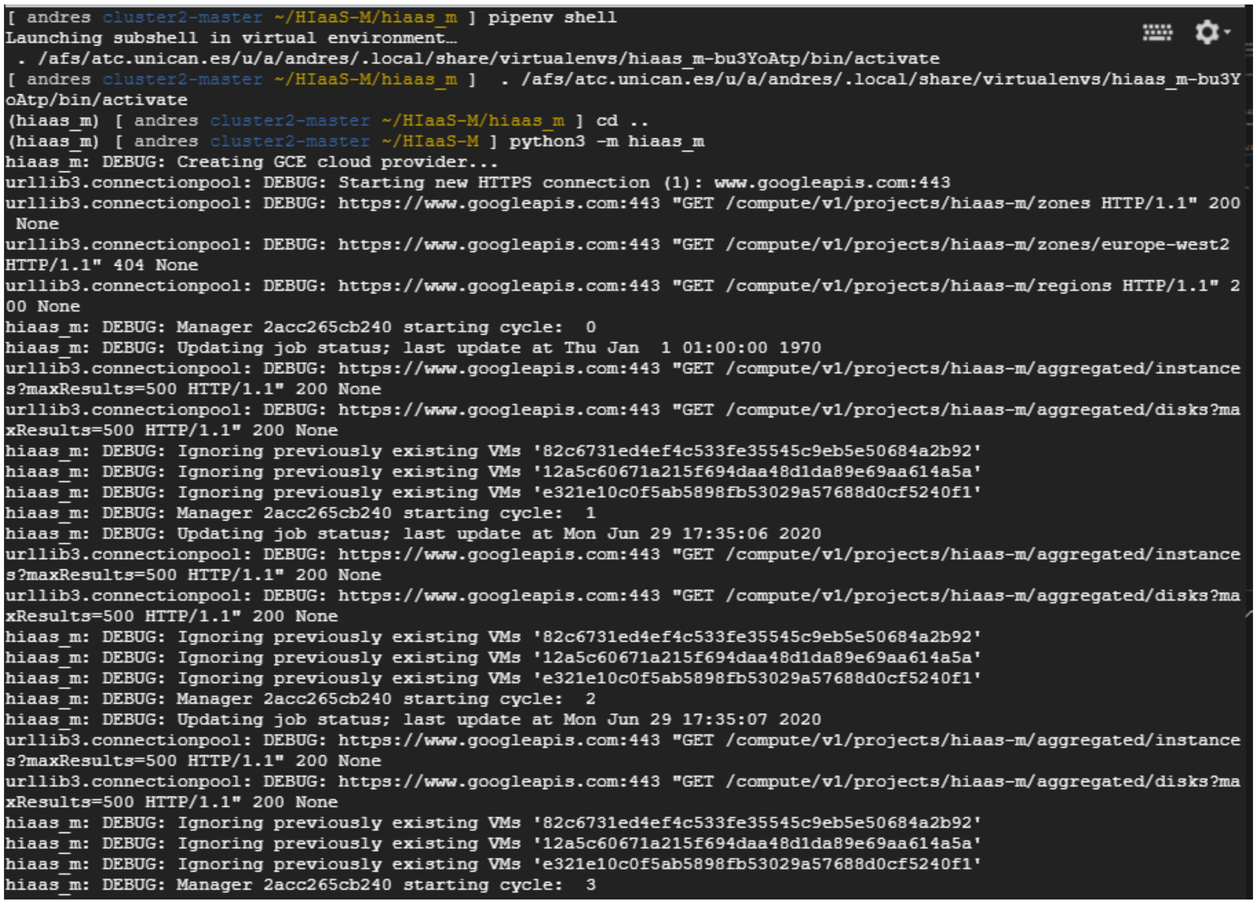
Figure 7.3 shows this process and the beginning of the debug output produced by the manager. This verbose output is generated because the debug option is activated in the configuration file. If there is no need for such amount of information, it is recommended to change this parameter to INFO. Leaving this configuration in a production environment is unnecessarily overwhelming and can even lead to expose vulnerabilities and/or private information.

7.3 Local execution of a job, on-premise

Even after the development of the framework, the on-premise infrastructure is intended to take care of any job execution, both local and remote. For this reason, the contributions of HIaaS-M are better appreciated understanding the functioning of the existing infrastructure for a local execution. This way it will be easier to understand the little disturbance that the new manager introduces to execute a job in the cloud and thus encourage its utilization.

Configuration and management (queuing, execution) to run a new in the cluster through OpenGE is performed through a bash script that facilitates the task. This script, shown in 7.4, is a quasi-standard among the research group at the university as most of the users will use it to configure their job executions by only changing the parameters inside it. The most important parameters are listed below.

- job_name: name of the job; identifier shown when the qstat command is executed
- email: email address for notifications of changes in job status: finished, error, etc.
- output_path: the relative path where the results of the execution will be stored.

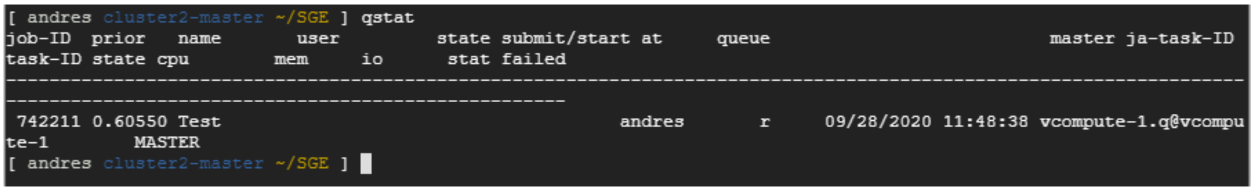


```
[ andres cluster2-master ~/HIaaS-M/hiaas_m ] pipenv shell
Launching subshell in virtual environment…
 . /afs/atc.unican.es/u/a/andres/.local/share/virtualenvs/hiaas_m-bu3YoAtp/bin/activate
[ andres cluster2-master ~/HIaaS-M/hiaas_m ]  . /afs/atc.unican.es/u/a/andres/.local/share/virtualenvs/hiaas_m-bu3Y
oAtp/bin/activate
(hiaas_m) [ andres cluster2-master ~/HIaaS-M/hiaas_m ] cd ..
(hiaas_m) [ andres cluster2-master ~/HIaaS-M ] python3 -m hiaas_m
hiaas_m: DEBUG: Creating GCE cloud provider...
urllib3.connectionpool: DEBUG: Starting new HTTPS connection (1): www.googleapis.com:443
urllib3.connectionpool: DEBUG: https://www.googleapis.com:443 "GET /compute/v1/projects/hiaas-m/zones HTTP/1.1" 200
 None
urllib3.connectionpool: DEBUG: https://www.googleapis.com:443 "GET /compute/v1/projects/hiaas-m/zones/europe-west2
HTTP/1.1" 404 None
urllib3.connectionpool: DEBUG: https://www.googleapis.com:443 "GET /compute/v1/projects/hiaas-m/regions HTTP/1.1" 2
00 None
hiaas_m: DEBUG: Manager 2acc265cb240 starting cycle:  0
hiaas_m: DEBUG: Updating job status; last update at Thu Jan  1 01:00:00 1970
urllib3.connectionpool: DEBUG: https://www.googleapis.com:443 "GET /compute/v1/projects/hiaas-m/aggregated/instance
s?maxResults=500 HTTP/1.1" 200 None
urllib3.connectionpool: DEBUG: https://www.googleapis.com:443 "GET /compute/v1/projects/hiaas-m/aggregated/disks?ma
xResults=500 HTTP/1.1" 200 None
hiaas_m: DEBUG: Ignoring previously existing VMs '82c6731ed4ef4c533fe35545c9eb5e50684a2b92'
hiaas_m: DEBUG: Ignoring previously existing VMs '12a5c60671a215f694daa48d1da89e69aa614a5a'
hiaas_m: DEBUG: Ignoring previously existing VMs 'e321e10c0f5ab5898fb53029a57688d0cf5240f1'
hiaas_m: DEBUG: Manager 2acc265cb240 starting cycle:  1
hiaas_m: DEBUG: Updating job status; last update at Mon Jun 29 17:35:06 2020
urllib3.connectionpool: DEBUG: https://www.googleapis.com:443 "GET /compute/v1/projects/hiaas-m/aggregated/instance
s?maxResults=500 HTTP/1.1" 200 None
urllib3.connectionpool: DEBUG: https://www.googleapis.com:443 "GET /compute/v1/projects/hiaas-m/aggregated/disks?ma
xResults=500 HTTP/1.1" 200 None
hiaas_m: DEBUG: Ignoring previously existing VMs '82c6731ed4ef4c533fe35545c9eb5e50684a2b92'
hiaas_m: DEBUG: Ignoring previously existing VMs '12a5c60671a215f694daa48d1da89e69aa614a5a'
hiaas_m: DEBUG: Ignoring previously existing VMs 'e321e10c0f5ab5898fb53029a57688d0cf5240f1'
hiaas_m: DEBUG: Manager 2acc265cb240 starting cycle:  2
hiaas_m: DEBUG: Updating job status; last update at Mon Jun 29 17:35:07 2020
urllib3.connectionpool: DEBUG: https://www.googleapis.com:443 "GET /compute/v1/projects/hiaas-m/aggregated/instance
s?maxResults=500 HTTP/1.1" 200 None
urllib3.connectionpool: DEBUG: https://www.googleapis.com:443 "GET /compute/v1/projects/hiaas-m/aggregated/disks?ma
xResults=500 HTTP/1.1" 200 None
hiaas_m: DEBUG: Ignoring previously existing VMs '82c6731ed4ef4c533fe35545c9eb5e50684a2b92'
hiaas_m: DEBUG: Ignoring previously existing VMs '12a5c60671a215f694daa48d1da89e69aa614a5a'
hiaas_m: DEBUG: Ignoring previously existing VMs 'e321e10c0f5ab5898fb53029a57688d0cf5240f1'
hiaas_m: DEBUG: Manager 2acc265cb240 starting cycle:  3
```

Figure 7.3: Starting HIaaS-M

```
# Launch script for simple jobs (sequential)
shell_name="/bin/bash"
# Name of the job
job_name="Test.cloud"
#Email address to which openGE will send the job status
email=phc24@alumnos.unican.es
# relative path of the output(-o) and error files (-e)
output_path="/afs/atc.unican.es/u/a/andres/SGE/results/out/"
error_path="/afs/atc.unican.es/u/a/andres/SGE/results/error/"
forced_queue="local.q"
# indicate if the job is: long, short or interactive and the group
# that describes the node's characteristics
execution_computer="$1"
# Send job to the openGE scheduler
qsub  << eof
#!/bin/bash
#$ -S $shell_name
#$ -N $job_name
#$ -o /dev/null
#$ -e /dev/null
#$ -v output_path=$output_path
#$ -v error_path=$error_path
#$ -q $forced_queue
#$ -V
#$ -M $email
#$ -m n
##$ -l $execution_computer
##$ -l h_vmem=4G
# SGEEE-AFS Permissions
kinit -k -t /home/Keytab/krb5.keytab.$USER $USER
aklog -c ATC.UNICAN.ES -k ATC.UNICAN.ES
# Job encapsulation
#
mkdir -p /scrath/\$USER/\$JOB_ID
if [ true ]; then
        stress -c 1 --vm-bytes 1G -t 300s
fi 1>\$output_path\$REQUEST.o\$JOB_ID 2>\$error_path\$REQUEST.e\$JOB_ID
eof
```

Figure 7.4: Requesting a job execution to OpenGE that will be executed locally



```
[ andres cluster2-master ~/SGE ] qstat
job-ID  prior   name       user         state submit/start at     queue                          master ja-task-ID
task-ID state cpu        mem     io      stat failed
-----------------------------------------------------------------------------------------------------------------
----------------------------------------------------
 742211 0.60550 Test                                   andres       r     09/28/2020 11:48:38 vcompute-1.q@vcompu
te-1        MASTER
[ andres cluster2-master ~/SGE ] 
```

Figure 7.5: Local job executing in on-premise execution VM vcompute-1

- error_path: the relative path where the error files, if any, will be stored.
- forced_queue: this is the added parameter to make the execution interact with the framework. It tells OpenGE in which queue the job must execute. The job will be placed in that queue even if there is no resources available in it, which is the desired behaviour, forcing it to wait until HIaaS-M provides a resource that is ready to execute the job, the cloud VM.
- execution_computer: This parameter tells to OpenGE in which execution (vcompute-) the job should execute. It doesn't override the forced_queue parameter, being the queue more important.

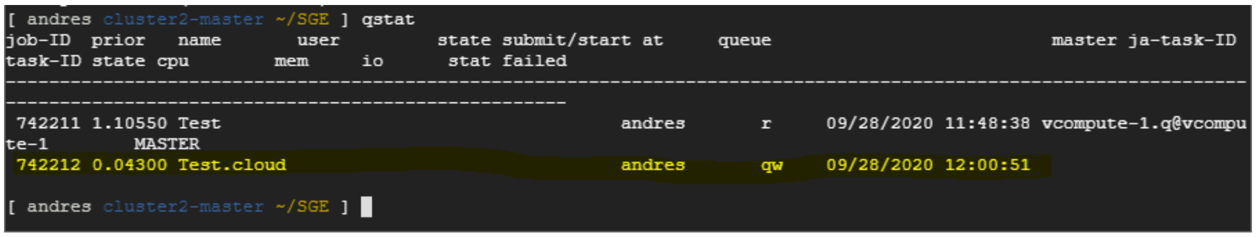
In figure 7.4 a typical configuration for a simple job execution can be seen.

For this demonstration, the most important part is the choice made by the user in the use of the forced_queue parameter as it determines in which queue the job should be executed and acts as a trigger for the framework. As a reminder, the static policy configured for the manager is that all the jobs that are NOT included in the 'cloud.q' queue must be executed locally and no action will be taken by the manager.

The script can be executed in several ways, for example by using the "sh sge.sh 1" command. After the script gets executed using this method, a message that reads: "Your job 742211 ("Test") has been submitted" appears on the screen. To check the state of the job, the qstat command will be used again, which output can be seen in figure 7.5. This output means shows that the job was running in the execution VM vcompute-1, as specified in the script, and now it is in a 'r' ready state.

7.4 Transparent execution of a job in the cloud

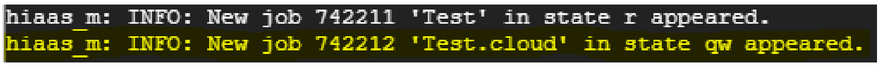
Our starting point will be the one described at the end of previous section. The cluster is currently running a job in on-premise hardware (See figure 10.5). As the manager constantly monitors OpenGE's queues (every minute) to find pending jobs, a new job will be manually inserted into the queue system using the same bash script as in previous section. From the user's point of view, the only change required is the name of the queue in which the job will be executed. To do so, the forced_queue parameter has to be changed from local.q to cloud.q and then the script is executed the same way as before.



```
[ andres cluster2-master ~/SGE ] qstat
job-ID  prior   name       user         state submit/start at     queue                          master ja-task-ID
task-ID state cpu        mem     io      stat failed
-----------------------------------------------------------------------------------------------------------------
---------------------------------------------------
 742211 1.10550 Test                                        andres       r     09/28/2020 11:48:38 vcompute-1.q@vcompu
te-1        MASTER
 742212 0.04300 Test.cloud                                  andres       qw    09/28/2020 12:00:51

[ andres cluster2-master ~/SGE ]
```

Figure 7.6: Job 'Test.cloud' in a waiting state in openGE



```
hiaas_m: INFO: New job 742211 'Test' in state r appeared.
hiaas_m: INFO: New job 742212 'Test.cloud' in state qw appeared.
```

Figure 7.7: Manager checking the jobs and detecting a new job that will execute in the cloud

The script output shows the same message: "Your job 742212 ("Test.cloud") has been submitted" exactly as before, nothing else has change to the user.

Once again, the state of the queues and jobs will be consulted using the qstat command which results can be seen in figure 7.6. This time, the job Test.cloud is labeled in a "qw" state, meaning that it is waiting for execution resources (the cloud VM in this case). The cloud VM will take some minutes to be fully functional and as soon as OpenGE detects an available node, the job will be sent there to be executed

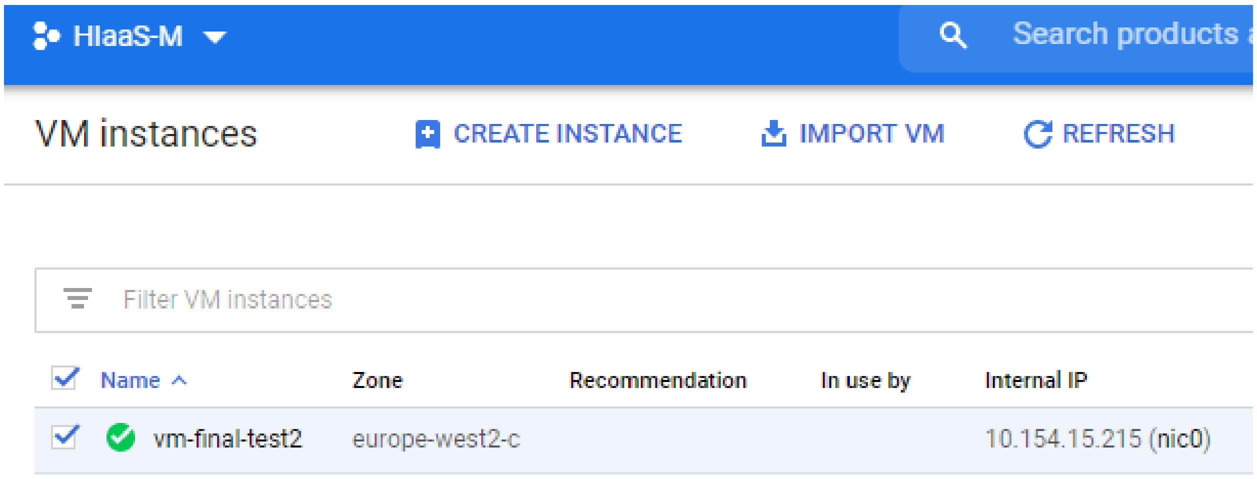
Concurrently, the manager has detected a new job that has to be sent to the cloud and will display an INFO level message as shown in figure 7.7. After the detection of this job, it will be included in the managed jobs and thus a VM in the cloud will be created.

If everything works correctly, by accessing the GCE console the creation process of a new VM can be observed. Once finished booting and setting up the necessary software, it will appear in the website interface as figure 10.8 shows. In this case the VM is called vm-final-test2 to distinguish it from the rest of them that are created with a name composed by 'cloud-vm-' plus a unique uuid.

After the VM finishes with its deployment, if in cluster2-master the command qconf -sel is executed, the execution hosts that openGE is managing can be seen. Figure shows the output of the command, showing vm-final-test2 as an additional execution host.

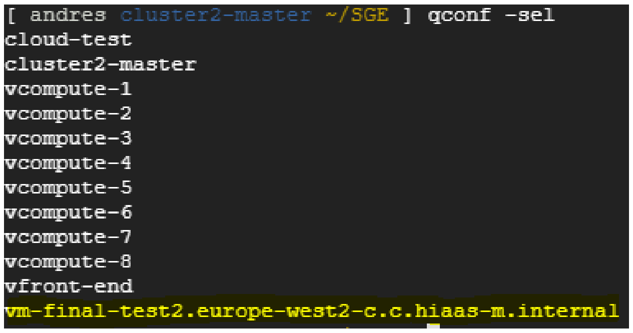
Having a free execution host in the 'cloud.q' queue, openGE can now send the job to it. To check this, the qstat command is executed again. As seen in figure 7.10, the output of the qstat command shows that the "Test.cloud" job is executing in the newly detected computing resource (the cloud VM).

However, it is important to notice that if, after the remote VM creation, the assigned job enters a undesired state (runtime errors, required resources not found, etc.), the framework (after a configured delay) will kill the cloud VM because it detects that there are no jobs running inside it, and that the VM is up and running for more time than the configured delay. This clean-up is done in every cycle of the manager class



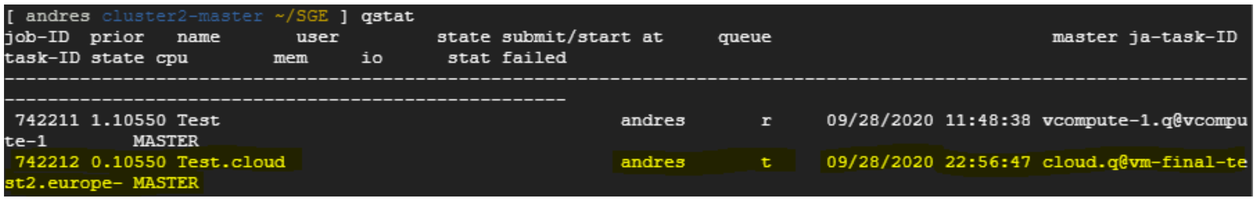
Name	Zone	Recommendation	In use by	Internal IP
vm-final-test2	europe-west2-c			10.154.15.215 (nic0)

Figure 7.8: A new VM appears in the GCE console where the job will execute



```
[ andres cluster2-master ~/SGE ] qconf -sel
cloud-test
cluster2-master
vcompute-1
vcompute-2
vcompute-3
vcompute-4
vcompute-5
vcompute-6
vcompute-7
vcompute-8
vfront-end
vm-final-test2.europe-west2-c.c.hiaas-m.internal
```

Figure 7.9: Execution hosts managed by openGE in cluster2-master



```
[ andres cluster2-master ~/SGE ] qstat
job-ID  prior   name       user         state submit/start at     queue                          master ja-task-ID
task-ID state cpu        mem     io      stat failed
-----------------------------------------------------------------------------------------------------------------
---------------------------------------------------
 742211 1.10550 Test                                         andres       r     09/28/2020 11:48:38 vcompute-1.q@vcompu
te-1        MASTER
 742212 0.10550 Test.cloud                                   andres       t     09/28/2020 22:56:47 cloud.q@vm-final-te
st2.europe- MASTER
```

Figure 7.10: Job executing in the cloud

and it applies to any VM that is found in such state. After the VM gets deleted, OpenGE will check if there are other available nodes to run the job and if not, the job will remain in a waiting state.

In summary, this chapter described a use case where, after having the framework running, a HPC/HTC user can send a test job to both on-premise and cloud infrastructures just by changing a parameter inside a script.

Chapter 8

Conclusions and future work

In this chapter, a summary of the work done, as well as ideas of where and how it could be improved will be shown.

At the date of conclusion of this thesis the software developed is capable of scheduling jobs in a transparent manner from the user point of view. Furthermore, due to the way HIaas-M was designed, defining abstract interfaces for the software pieces that interact with it (cloud providers, job schedulers, etc), it is able to achieve great modularity, making it feasible to extend the framework to work with alternative cloud providers (AWS, Microsoft Azure) and schedulers (Slurm, NFS), following the schema used for GCE and OpenGE.

Taking into account that many entities have on-premise infrastructure, HIaaS-M could be of great value to many of them, especially in the academic field where budgets may be low, and the frequent experiments demand new and modern computation resources that are hard to find, particularly for small research groups.

As an example, thanks to this framework, a research group could access to unlimited computing resources or to extremely expensive technologies for short periods of time (only the required to complete a proposed experiment), paying only for what they need and avoiding the acquisition costs of such hardware.

Similarly to any piece of software, improvements can be made, especially in the generalization of the system, creating a common interface that unifies several job schedulers and providers, something similar to the growing trend of MultiClouds [35].

There is also room for improvement in the implementation of policies that decide when to create or destroy a VM as well as when a job should be consider a good candidate to be executed in the cloud. These methods can get really complicated as advanced techniques like reinforced learning could be used to analyze the workload of a cluster and to predict when new cloud infrastructure is needed.

Another area of improvement could consist on the automatization of the decision about the size and characterization of the VM to create. This topic is not trivial and could easily lead to another interesting research. Further research needs to be done in the use of other schemes of architecture in the cloud like, e.g.

the use of containers (docker, kubernetes) and how it would impact in the performance, energy consumption and money saving of it.

Bibliography

[1] *3Mares - University of Cantabria*. 2020. URL: https://www.ce.unican.es/resource/computing/.

[2] *Advantages of using AFS*. 2010. URL: https://uit.stanford.edu/service/afs/intro/whyafs#:~:text=Better%20Networking%20Performance%3A%20AFS%20was,the%20update%20to%20the%20server..

[3] *Advantages of using AFS*. 2010. URL: https://uit.stanford.edu/service/afs/intro/whyafs#:~:text=Better%20Networking%20Performance%3A%20AFS%20was,the%20update%20to%20the%20server..

[4] *Altamira - Institute of physics of cantabria*. 2020. URL: https://ifca.unican.es/en-us.

[5] *Amazon AWS*. 2020. URL: https://aws.amazon.com.

[6] Jeff Bar. *Cloudbursting – Hybrid Application Hosting*. 2008. URL: https://aws.amazon.com/blogs/aws/cloudbursting/.

[7] Antonio Celesti et al. "How to enhance cloud architectures to enable cross-federation". In: *2010 IEEE 3rd international conference on cloud computing*. IEEE. 2010, pp. 337–345.

[8] Yan Cui et al. "Total cost of ownership model for data center technology evaluation". In: *2017 16th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*. IEEE. 2017, pp. 936–942.

[9] *Docker containers*. 2020. URL: https://www.docker.com/resources/what-container.

[10] Cameron Fisher. "Cloud versus On-Premise Computing". In: *American Journal of Industrial and Business Management* 08 (Jan. 2018), pp. 1991–2006. DOI: 10.4236/ajibm.2018.89133.

[11] Flexera. *Flexera 2020 State of the Cloud Report*. 2020. URL: https://info.flexera.com/SLO-CM-REPORT-State-of-the-Cloud-2020.

[12] Ian Gable et al. "A batch system for HEP applications on a distributed IaaS cloud". In: *Journal of Physics: Conference Series*. Vol. 331. 6. 2011.

[13] *Gitlab official website*. 2020. URL: https://gitlab.com.

[14] *GNU Bash*. 2017. URL: http://www.gnu.org/software/bash/.

[15] Google. *Google Compute Engine prices*. 2020. URL: https://cloud.google.com/compute/vm-instance-pricing.

[16] *Google cloud computing*. 2020. URL: https://cloud.google.com/.

[17] *Google cloud storage princing*. 2020. URL: https://cloud.google.com/storage/pricing.

[18] Abhishek Gupta et al. "Evaluating and Improving the Performance and Scheduling of HPC Applications in Cloud". In: *IEEE Transactions on Cloud Computing* (Aug. 2014). DOI: 10.1109/TCC.2014.2339858.

[19] *How puppet works*. 2016. URL: https://docs.oracle.com/cd/E53394_01/html/E77676/gqqvw.html.

[20] IIETF. *The OAuth 2.0 Authorization Framework*. RFC 6749. Internet Engineering Task Force (IETF), Oct. 2012. URL: https://tools.ietf.org/html/rfc6749.

[21] Kodguru. *Puppet-module-afs*. 2020. URL: https://github.com/kodguru/puppet-module-afs.

[22] *LDAP protocol*. 2020. URL: https://www.ldap.com/.

[23] *Microsoft Azure*. 2020. URL: https://azure.microsoft.com.

[24] *OpenLDAP website*. 2020. URL: https://www.openldap.org/.

[25] *OpenVPN*. URL: https://openvpn.net/private-tunnel/.

[26] Oracle. *BEGINNER'S GUIDE TO SUN™ GRID ENGINE 6.2*. 2018. URL: http://moo.nac.uci.edu/~hjm/Sun_Grid_Engine_62_install_and_config.pdf.

[27] *Puppet*. 2020. URL: https://puppet.com.

[28] relleums. *Qstat parsing module*. 2020. URL: https://github.com/relleums/qstat.

[29] Slurm. *Slurm Scheduler*. 2020. URL: https://slurm.schedmd.com/.

[30] *Slurm cloud bursting capabilities*. 2020. URL: https://slurm.schedmd.com/elastic_computing.html.

[31] StackOverFlow. *StackOverFlow Developer Survey Results*. 2019. URL: https://insights.stackoverflow.com/survey/2019.

[32] Univa. *Univa workload manager*. 2020. URL: https://www.univa.com/products/univa-grid-engine.php.

[33] California University. *Running a Job on HPC using Slurm*. 2019. URL: https://hpcc.usc.edu/resources/documentation/slurm/.

[34] *What is a CDN*. 2020. URL: https://www.cloudflare.com/en-gb/learning/cdn/what-is-a-cdn/.

[35] *What is multicloud*. 2019. URL: cloudflare.com/learning/cloud/what-is-multicloud/.

[36] *What is supercomputing*. 2020. URL: https://www.hpe.com/us/en/what-is/supercomputing.html.

[37] *What is the Total Cost of Ownership*. 2020. URL: https://www.thebalancecareers.com/total-cost-of-ownership-tco-2276009.

[38] *Windows Subsystem for Linux (WSL)*. 2020. URL: https://docs.microsoft.com/es-es/windows/wsl/about.

[39] Hao Wu et al. "Automatic cloud bursting under fermicloud". In: *2013 International Conference on Parallel and Distributed Systems*. IEEE. 2013, pp. 681–686.

Appendices

Appendix A

HIaaS-M inside the system

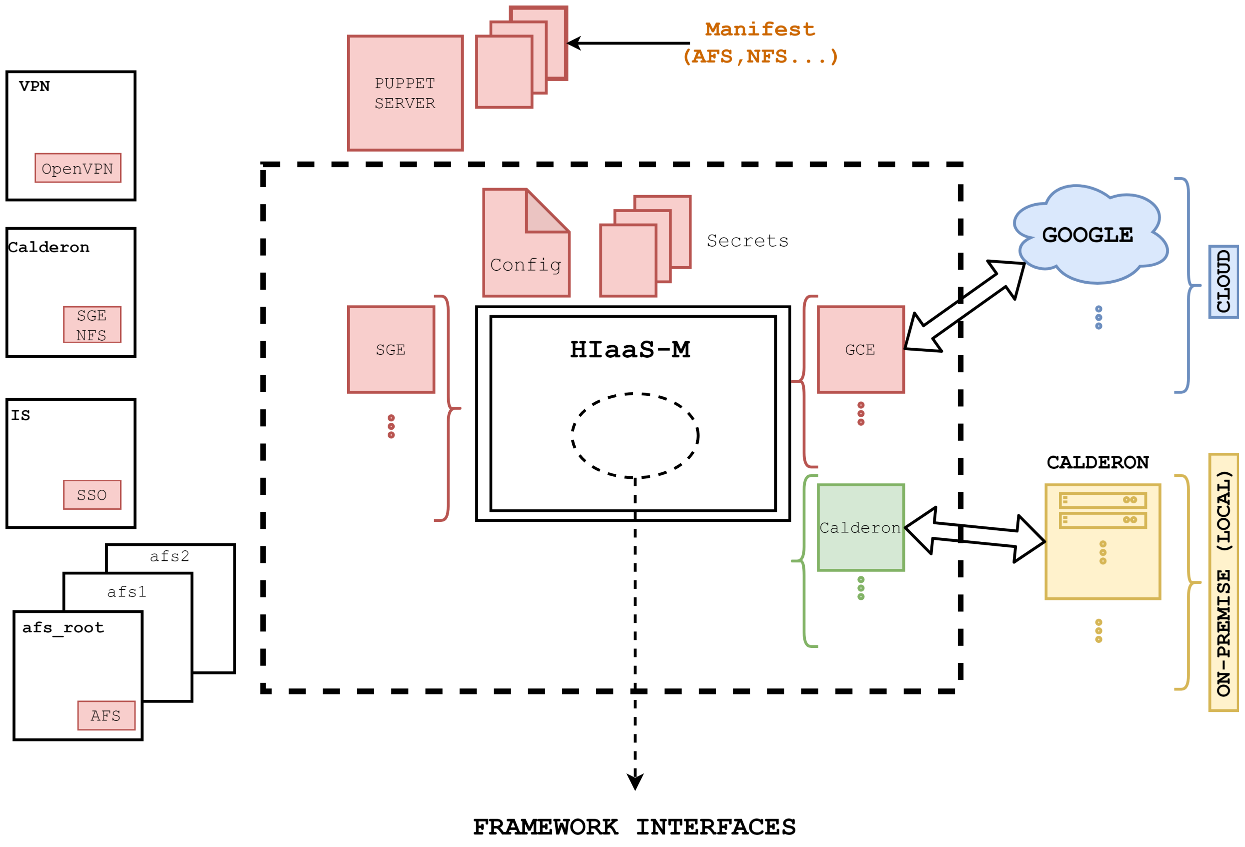


Figure A.1: Where HIaaS-M locates inside the whole system

Appendix B

HIaaS-M UML diagram

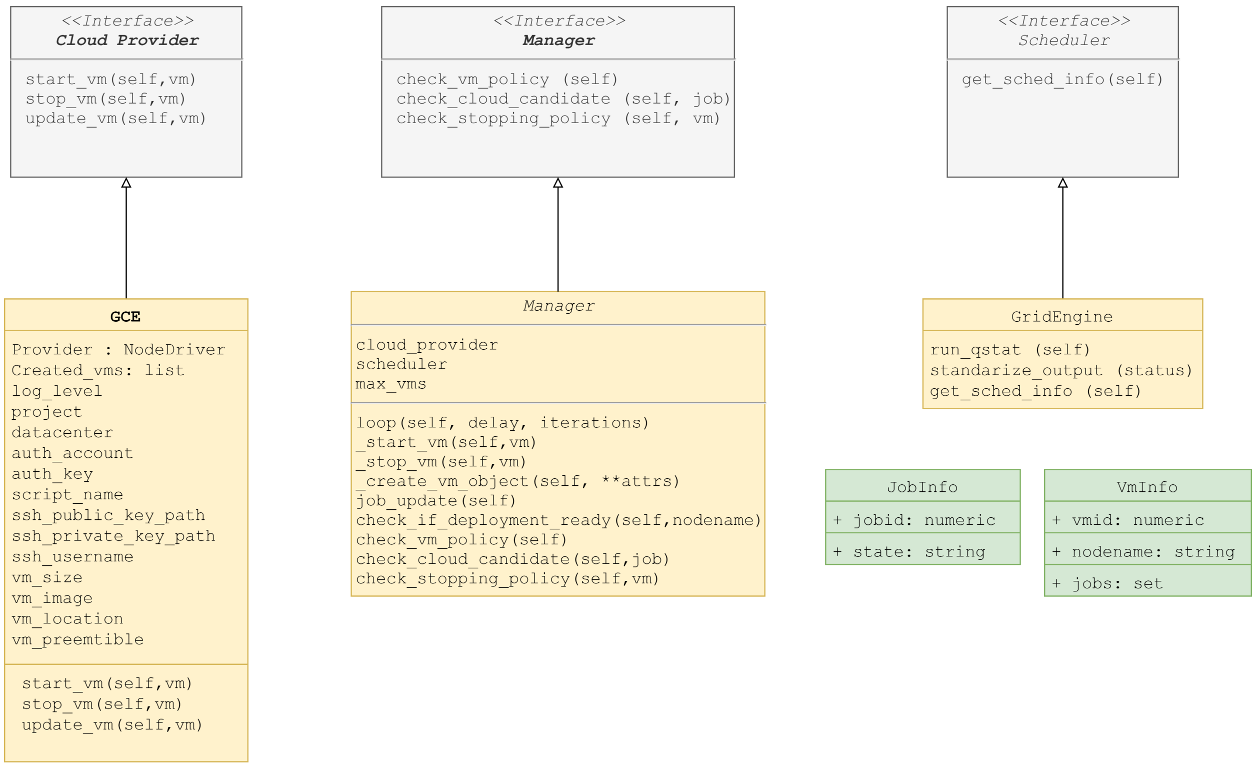


Figure B.1: HIaaS-M UML diagram