MASTER IN INNOVATION AND RESEARCH IN INFORMATICS
SERVICE ENGINEERING

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BARCELONATECH

MASTER'S THESIS

DATA ANALYTICS AS A SERVICE:
A LOOK INSIDE THE PANACEA PROJECT

CRISTOBAL YANUARIO SOLIS PATRON

DIRECTOR:

M. ANA JUAN FERRER – ATOS RESEARCH & INNOVATION SPAIN

CODIRECTOR:

Dr. DAVID GARCIA PEREZ – ATOS RESEARCH & INNOVATION SPAIN

'PONENT':

Dra. MARIA RIBERA SANCHO – DEPARTMENT OF SERVICE AND INFORMATION SYSTEM ENGINEERING

DEFENSE PERIOD: JULY 6th-10th 2015

ACKNOWLEDGMENTS

This thesis is the result of hard work but also from great help that came from diverse sources. I would like to take this space to thank very briefly before presenting this document.

First and probably one of the most important persons to thank is my mom. While she is not here with us anymore, we made this sacrifice together, she gave me all her strength and push me to my limits. I am really grateful for all you have done, mom and I know that from wherever you are, you are proud that I made it this far on my own.

My dad also played a decisive role in this master degree by giving his continuous support in all ways during this journey, either economically or morally even from a distance.

Of course, thanks for my family. They helped me and more importantly my mom until the last moments and they also respected our decisions regarding this master's degree and therefore this thesis.

To my director and codirector, for having enough patience with me and helping me with everything I needed at times. Without their support I could not have been able to deliver this document.

I also want to thank my ponent, for her professionalism and time following my thesis even when she already got enough work in her hands. Thanks for the support and insights provided along the way.

Thanks to all the people and friends I have met along the way in this journey. For their courageous words, their scolding and also their support whenever I needed it.

Last but not least I want to thank the "Consejo Nacional de Ciencia y Tecnología" (CONACYT) of my home country, Mexico; for its scholarship without which I could not have done and finish this master degree. A special thanks to all the good people in Mexico who pay their taxes (including myself) since it is the fuel that makes CONACYT's scholarships exist and continue. The good, are more.

ABSTRACT

The Cloud… The evolution of computing, this has been the hot topic for quite some time; a few years have passed since the beginning of what is commonly known as Cloud computing. During all this years the Cloud has evolved from a few thousand machines to enormous datacenters that interconnect with each other in diverse regions to provide the power required for nowadays connected society. While it is currently used by the general public, Cloud computing has become a norm within enterprise IT. The competition among public Cloud providers is red hot; private Cloud continues to grab increasing shares of IT budgets, and hybrid Cloud strategies are beginning to conquer the enterprise IT world. However, even if it has evolved, the Cloud continues to be physical computers interconnected by physical networks in a physical space; the Free Software Foundation Europe (FSFE) puts it simply: "There is no Cloud, just other people's computer". [1]

In such setting, PANACEA is born as a solution to the complete set of problems that the Cloud may encounter during its full working life cycle, since the very first use to the continuing grow and decrease through time. The main objective of the project PANACEA is to provide Proactive Autonomic Management of Cloud Resources, which is why it is called PANACEA, in short. The aim is to set a proactive autonomic management of Cloud resources based on machine learning. It presents the main techniques to be used to achieve this goal: two-level architecture, middleware at each node of the overlay network and the virtualization framework.

PANACEA is being developed by a consortium of different stakeholders across Europe and Middle East, in this context a series of use cases have been developed by said consortium to ultimately test and verify that the goal of PANACEA is being achieved. Those use cases cover an array of different services and applications in which the cloud is used or could be used in the near future. One of such cases is that of Data Analytics as a Service (DAaaS). Data Analytics or even known as Big Data Analytics is the process of examining large data sets containing a variety of data types, i.e., big data, to uncover hidden patterns, unknown correlations, market trends, customer preferences and other useful business information. The analytical findings can lead to more effective marketing, new revenue opportunities, better customer service, improved operational efficiency, competitive advantages over rival organizations and other business benefits. This kind of process requires a high amount of resources that only the Cloud is able to provide, but as more and more resources are required and the more and more people are using it the probabilities of encountering an issue, big or small, increases drastically. PANACEA aims in this way to provide the availability and performance required by this use case by introducing a set of innovations described in this work.

In this particular work we will take a close look to two of these innovations in the DAaaS use case and the steps need to perform and validate said innovations. In it, we review different alternatives to our use case and/or the innovations we are introducing as part of PANACEA, we describe part of what PANACEA is and what the DAaaS use case is composed of and finishing with the steps and testes performed to validate the innovations in the use case. The cornerstone of all of the work described before is making elasticity possible for Hadoop in Cloud; elasticity is the capacity for a system (or a particular cloud layer) autonomously adapts its capacity to workload over time. To be able to do this we performed a study to understand how to make it possible for Hadoop to increase and decrease its

resources so then we can then make it possible for Hadoop to do it autonomously; the tests we mentioned before aim to give us enough evidence that we have done it correctly and we have reached the objective of making Hadoop elastic in Cloud. Please bear in mind that the work captured here was done as part of an internship that lasted 4 months, reader discretion is advised.

Table of Contents

INDEX OF FIGURES

GLOSSARY

Cloud computing: Cloud computing refers to the practice of transitioning computer services such as computation or data storage to multiple redundant offsite locations available on the Internet, which allows application software to be operated using internet-enabled devices. Clouds can be classified as public, private, and hybrid.

AWS: Amazon Web Services, Amazon Web Services is a collection of remote computing services, also called web services that make up a cloud computing platform offered by Amazon.com. These services are based out of 11 geographical regions across the world.

EMR: Elastic Map Reduce, Amazon Elastic MapReduce (Amazon EMR) is a web service that simplifies big data processing, providing a managed Hadoop framework that distribute and process vast amounts of data across dynamically scalable Amazon instances. It can also run other popular distributed frameworks such as Spark and Presto in Amazon EMR, and interact with data in other AWS data stores such as Amazon S3 and Amazon DynamoDB.

DAaaS: Data Analytics as a Service, Data Analytics as a Service is an internal name for the aimed service to be delivered by Atos using PANACEA and the use case reviewed in this document. The counterpart in the nowadays market will be that of big data analytics, which refers to the process of collecting, organizing and analyzing large sets of data to discover patterns and other useful information.

HDFS: Hadoop Distributed File System, The Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop applications. It is a distributed file system that provides high-performance access to data across Hadoop clusters. Like other Hadoop-related technologies, HDFS has become a key tool for managing pools of big data and supporting big data analytics applications.

YARN: Yet Another Resource Negotiator, Apache Hadoop YARN (short, in self-deprecating fashion, for Yet Another Resource Negotiator) is a cluster management technology. It is one of the key features in second-generation Hadoop.

SLA: Service Level Agreement, A **service level agreement** (**SLA**) is a contract between a service provider (either internal or external) and the end user that defines the level of service expected from the service provider. SLAs are output-based in that their purpose is specifically to **define** what the customer will receive.

VM: Virtual Machine, A **virtual machine** (**VM**) is an operating system OS or application environment that is installed on software which imitates dedicated hardware. The end user has the same experience on a **virtual machine** as they would have on dedicated hardware.

IaaS: Infrastructure as a Service, in an IaaS model, a third-party provider hosts hardware, software, servers, storage and other infrastructure components on behalf of its users. IaaS providers also host users' applications and handle tasks including system maintenance, backup and resiliency planning.

PaaS: Platform as a Service, in it, cloud providers deliver a computing platform, typically including operating system, programming language execution environment, database, and web server. Application developers can develop and run their software solutions on a cloud platform without the cost and complexity of buying and managing the underlying hardware and software layers.

SaaS: Software as a Service, describes any cloud service where consumers are able to access software applications over the internet. The applications are hosted in centrally and can be used for a wide range of tasks for both individuals and organizations. It is sometimes referred to as "on-demand software" and is usually priced on a pay-per-use basis or using a subscription fee.

GFS: Google File System (GFS) is a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients; HDFS is based on it.

ML: Machine Learning, is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that can teach themselves to grow and change when exposed to new data. It also explores the construction and study of algorithms that can learn from and make predictions on data.

QoS: Quality of Service, It is defined as the overall performance of a telephony or computer network, particularly the performance seen by the users of the network, this definition is also applicable to any service (Infrastructure, Platform, etc.) in Cloud as it requires the Internet (the network of networks) to be able to provide said services.

CHAPTER 1 - INTRODUCTION

This document represents the master thesis on the Master in Innovation and Research in Informatics done at the Barcelona School of Informatics (Facultat d'Informàtica de Barcelona) of the Polytechnic University of Catalonia (Universitat Politècnica de Catalunya). It has been done as part of an internship in the company ATOS Spain SAE as part of the team in Atos Research & Innovation during the period from March to June 2015.

1.1 Context of the Work

1.1.1 Cloud computing and MapReduce

The use of Cloud [1] nowadays is more prolific than ever [2], more than likely we use it in some form and it is also possible that many of the services that we use in a daily basis is supported by a Cloud infrastructure in some form. The main categories for Cloud services are Infrastructure as a Service (IaaS) which includes such offerings as virtual server space, network connections, bandwidth, IP addresses and load balancers, Software as a Service (SaaS) often referred to as software-on-demand and utilizing it is akin to renting software rather than buying it and Platform as a Service (PaaS) providing a platform and environment to allow developers to build applications and services over the internet but these categories are by no means the only services that can be found in the Cloud and there are probably not the only ones that will come in service in the days to come.

One of the more recent examples for new services in the Cloud is that of Big Data Analytics but before explaining what is all about we need to understand just what Data Analytics is and how is it done in a normal basis and how we can port it to the Cloud to be offered as a Service. Data analysis or analytics is the process of finding the right data to answer a given question, understanding the processes underlying the data, discovering the important patterns in the data, and then communicating the results to have the biggest possible impact. This, of course, is done generally using diverse tools for extracting, processing and presenting the data. Data analytics is usually more efficient as the amount of data available for analysis is bigger but this has the drawback of being more difficult to process efficiently and in time for being useful, and therefore expert tools are needed.

A few examples of expert tools are Google File System paper in 2003 [3] and the MapReduce in 2004, [4]. The Google File Sytem (GFS) is a distributed file system developed by Google designed to provide efficient, reliable, access to data using large clusters of commodity hardware while the MapReduce is a programming model and an associated implementation for processing and generating large data sets; Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. Using these two technologies, different alternatives for the analysis of big data have sprung; the most notable example of it is Hadoop. The core technologies of Hadoop are MapReduce which is its most notable trait as well as the cleverly named Hadoop Distributed File System (HDFS) which is similar to GFS as it is specially designed for commodity hardware and its Yet Another Resource Manager (YARN) which is the most important component in Hadoop 2.x and in charge of everything MapReduce-related that is sent to the Hadoop cluster. All in all, Hadoop is known for being a reliable framework capable of doing a diverse range of tasks over huge amounts of data so they can be used later for specific

analysis. This capacity has been exploited by diverse companies including Yahoo, Facebook, Intel and others. In the study found in [5], we are presented with the reality that Hadoop can be executed in the Cloud and in some settings can even outperform its physical counterparts, this has also been evaluated in [6]; this gives more viability to the use of Hadoop in the Cloud to be used as the key for DAaaS.

1.1.2 The PANACEA project

Without a doubt, all the Cloud services have inherent problems due to the reality that everything in the Cloud have either a physical component (servers, switches, or other) or a software component and as such are prone to failure. These failures can appear anytime and anywhere and for various reasons, such failures can include memory leaks, I/O data failures, memory outage, network connectivity, etc. as well as human error issues or attacks to any Cloud infrastructure; therefore, it is needed that specific recovery and monitoring mechanisms are in place to guarantee the correct operation of such services. As of today, present Cloud computing platforms do not support redundant, self-recovering programming models for recovering from many inevitable hardware/software failures, hence the need of a set of technologies that can provide such recovering capabilities.

The PANACEA project [7] is, in this way, born as a join initiative for providing Proactive Autonomic Management of Cloud Resources, based on Machine Learning (ML), as a remedy to the problems mentioned above. Many Cloud platforms, as we will see across this work, are capable of recovering from such problems, however they do it in a reactive way thus meaning that until the problem arises it is dealt with. PANACEA therefore aims to provide solutions to such problems in a proactive way meaning that it can control a situation by responding and taking measures before the problem may happen; as we mentioned before, by using ML PANACEA can continually monitor looking for any sign of imminent failure and with a set of rules modify the infrastructure to overcome the detected issue without impacting severely in the quality of service (QoS) of the services in Cloud.

The proposed solutions presented by PANACEA are very ambitious and of course need to be validated using or presenting use cases that provide evidence of the success and functionality of said solutions. In this context, DAaaS is one of the use cases devised by the consortium in charge of PANACEA as it have been identified to help validate the outputs needed to reach the goal of the project. The idea behind of the DAaaS use case is that of providing a platform where there can be configurable Big-Data services in which users/companies can bring their own data to be processed or that third parties can offer their curated data sources (e.g. demographic data provided by governments) and/or algorithms preconfigured for processing data. As a use case for PANACEA validation, it will be conformed of a cluster of Hadoop, a monitoring solution that will help us understand what is happening under the hood and a Cloud manager to be as close as possible to the final implementation of a DAaaS over PANACEA; more information about the technologies, architecture and goals for this use case will be commented later in this work.

1.2 Motivation

While there may be evidence of some companies or alternatives that are already using or providing something similar to what is expected will be the result of PANACEA, the main motivation for the project as it can be consulted in [7] is to provide such solutions in an autonomous and proactive manner. It is known that nowadays Cloud computing platforms lack of technologies for recovering from many inevitable hardware/software failures, at least in an autonomous way not even

proactive. PANACEA aims to do so, as said before, by providing innovations based on Machine Learning; such innovations will allow users several advanced possibilities in the Cloud in which a DAaaS platform is included. The DAaaS platform as well as other Cloud services will indeed benefit of such innovations that expect benefits both for the users as well as the providers of the Cloud services.

However, it is too ambitious to implement or even test all of the PANACEA innovations in the period of work that extends only to four months. In this short period of time, not even all of the objectives of the use case at hand (DAaaS) can be reached and as such the expected results will only reflect part of what this use case is aimed to do. In this way, the principal motivation for this work is to research how we can provide elasticity in the Cloud for the Hadoop implementation in the DAaaS use case.

1.3 Objectives

The main objective of this thesis as mentioned above is to showcase the benefits of PANACEA for one specific use case known internally as Data Analytics as a Service (DAaaS). The work will fully concentrate on **providing elasticity to Hadoop in Cloud**. Elasticity in the Cloud, as defined in [8], is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible. To do so, we firstly need to understand how can we adapt Hadoop to make it elastic, in which moments do we need to make it provision or de-provision resources as well as create rules and actions that need to be executed with the knowledge gathered. In this regard and to be able to provide the elasticity we desire for Hadoop a complete set of tasks were performed through this thesis to achieve such objective and are presented below:

- Adaptation of a series of algorithms of MapReduce developed for Hadoop 1.0 and makes them compatible with Hadoop 2.x: this will help to showcase the services available in the DAaaS use case for the potential users. It will also be able to improve the novelties of the PANACEA project by letting us understand what happens when a series of users are concurrently using the platform.
- Development of the images (RAW or QCOW2) that will serve as the backbone of a Hadoop cluster and make sure they are able to be deployed using OpenNebula; a software intended for the comprehensive management of virtualized data centers.
- Use of said algorithms and images for testing to showcase and guarantee that the services for DAaaS work as expected and that the ubiquitous monitoring and autonomic management execute correctly and work as intended.
- Perform a study about which parameters could be used to evaluate the Hadoop cluster health and saturation as well as developing some metrics probes based on the finding from the study. Said metrics probes will need to be able to communicate with the set of tools provided by PANACEA.
- Based on the aforementioned study and probes, develop a series of rules and actions that should engage whenever some conditions are met (to serve as an initial point for the autonomic management).
- Once everything is completed and tested, all of the images created for this objective should be able to be used as a starting point for the correct deployment of a Hadoop cluster usable for the DAaaS Use Case of PANACEA.

1.4 Organization

For the rest of the work, the organization of it is as follows. We will dedicate Chapter 2 for the state of the art using a literature review for academia and commercial alternatives to our use case of DAaaS as well as the tools needed for proactive and autonomic usage of Cloud services, at the end of such chapter a small discussion of the findings will take place. Chapter 3 is then focused on explaining a bit about what PANACEA is, the DAaaS scope inside the project, the use case used for validation and the approach used for achieving such scope. Chapter 4 summarizes the main work performed for this thesis including the infrastructure used for testing, how we build it and the organization of the tests performed, setting the ground for the chapter dedicated to the results of said tests. Chapter 5 goes in deep with the tests and provides the results used for validating that the scope of the work presented in Chapter 3 is reached and in line with what is expected while also give a brief of lessons learned during the course of this work. Finally, Chapter 6 will give the conclusions for this work and some future work that could be done to increase the benefits perceived by DAaaS and/or anything that could have been done but due to time constraints could not be implemented.

CHAPTER 2 - STATE OF THE ART

Data analytics or data analysis is, as the Cambridge dictionary defines, the process of examining information, especially using a computer, in order to find something out or to help with making decisions. A compliment to this definition provided by John Hopkins University in what of its courses say that "it is the process of finding the correct data to answer your question understanding the processes underlying the data, discovering the important patterns in the data, and then communicating your results to have the biggest possible impact". There are several tools and alternatives present nowadays that help us analyze the incredible and sometimes even ludicrous amount of data available for us, either as an individual or as a company; tools like Facebook Insights or the Google Analytics platform provide everyone with a set of tools that were not available just a few years ago. But, what happens when the information we need comes from within the company we work or we need to compare them with real data given by organizations around the world like government or ONGs? Or better yet, what if we need to "crunch" all the data we can gather and in this way have a more, better and clearer image of what data is telling us or what answers can it gives to us.

As we just saw there are a some scenarios where the amount of data available to us is so big that making sense through it is a titanic task, in such scenarios the usual approximation is to use "Big Data Analytics", this technology is basically performing data analysis but using big data as its source for finding useful information that can be used for making better decisions; Big data is a broad term for data sets so large or complex that traditional data processing applications are inadequate.

One popular way to fix the need for this kind of technology is to set up an owned analytic platform with our own hardware and software. However, this kind of platform is generally costly and even if we can acquire a cheaper platform, the speed that it can provide us will be more than slow for the purposes we define for it. In such cases the need for another technology such as Cloud computing, i.e. the delivery of on-demand computing resources over the Internet on a pay-for-use basis, is needed. Using Cloud computing, any company or organization can have access to the computation power need to use big data analytics and deliver results in a space of time where they are valid and useful.

In such context, PANACEA aims to usher enough innovations to provide better tools for the Cloud providers to be able to deliver better and faster Cloud infrastructures as well as more services available that will benefit the users of said Cloud providers. Such services could include our proposed Use Case of PANACEA, DAaaS, or many others. While PANACEA will benefit greatly both providers and clients, there are undoubtedly some commercial alternatives that offer some of the services that PANACEA aims to improve, such as DAaaS, as well as other improvements done in academy. Therefore it will be the aim of this chapter to present the different alternatives that are not only available but relevant to our case, in particular we go through Hadoop in Cloud offering concentrating specifically in their elasticity capabilities, monitoring options that provide tools for metrics collection and preferably some automatic actions that could benefit elasticity for Hadoop; we also include some commercial tools that can help automate the elasticity of Hadoop and some interesting projects from academia that could be used in any of the 3 different types of options given by commercial alternatives portrayed in this research.

2.1 Hadoop in the Cloud

While the main objective of PANACEA is of course propose something unique in the Cloud not only for DAaaS but for Cloud services, during this section, we will discuss various alternatives that provide only the DAaaS use case of PANACEA, more specifically we will look into alternatives to Hadoop in the Cloud which is the basic format of the DAaaS use case.

2.1.1 Amazon Elastic MapReduce

Amazon is a company well known for its Cloud services offering, Amazon Elastic MapReduce (EMR) is their offer of Hadoop in the Cloud inside the Amazon Web Services (AWS). In its webpage found in [9], Amazon explains its offer as a web service that makes it easy to quickly and cost-effectively process vast amounts of data. Its aim is to simplify the big data processing; it mainly uses Hadoop for big data processing although other alternatives and tools such as Spark and Presto can also be used in conjunction. The most prominent features to compare, as said by its webpage, include the following:

- Elastic: Amazon claim to provide up to thousands of instances to process data of any scale. It can increase and decrease **easily**.
- Reliable: Amazon EMR has tuned its Hadoop instances especially for the Cloud and therefore there seems to be no need for tuning. It monitors the clusters – retrying failed tasks and **automatically** replacing poorly performing instances.
- Flexible: Full access to instances to customize or install **multiple** Hadoop distributions and applications.

Rummaging through the documentation of its services there is no indication of how they do the automatic part of replacing poorly performing instances. The provisioning part is “easily” done manually using the console provided by Amazon, but other options including the EMR command line interface, SDKs or APIs. The monitor part uses two main actors for metric and alarms for users, the Amazon CloudWatch can be used to set alarms to warn the user when some metric go outside given parameters, this may include the data utilization and if some instances is running idle as well as the general health of the cluster, other tool for monitoring that can be used in Amazon EMR is an open source tool known as Ganglia that can be installed in any instance from the beginning and also include metrics and can generate reports and view the performance of the clusters or individual nodes, more info about Ganglia will be discussed later in this chapter.
Evidence of the use of Amazon EMR as a Hadoop in Cloud platform or part of it can be found in [10] and [11]. The main issue with Amazon EMR, is that it mainly consists of a static cluster it of course can be elastic but it is in need of user-defined metrics or instructions to grow or de-grow meaning that such rules do not exist, it will basically means that the cluster will have a fixed and static size. While a popular solution as it is, it is constrained to the tools available to it and also comes with an extra tag price to be available to use in the AWS.

2.1.2 OpenStack Sahara

Formerly known as Savanna but changed due to potential trademark issues, it is the elastic MapReduce alternative from OpenStack, probably one of the most known and widely used open source software for creating and provisioning private and public Clouds. As its homepage, found in [12], says the OpenStack can be managed through a dashboard (UI) or via the OpenStack API and has a booming ecosystem that offers OpenStack-powered products and services in its marketplace;

there is no doubt that in such ecosystem and marketplace there exist a place for DAaaS alternatives and as such OpenStack needs to be prepared and its answer to it is the Sahara project.
The Savanna project [13], for its part, aims to provide users of OpenStack with simple means to provision a Hadoop cluster at OpenStack by specifying several parameters like Hadoop version, cluster topology, nodes hardware details and a few more. Once such information is provided, Sahara will deploy the cluster, being able also to scale provisioned ones by adding or removing worker nodes on demand; however, this 'on demand' feature is only available by user intervention, the user will define the number of needed machines on the cluster and then Sahara will automatically make the operations required. There is actually a mention to 'Analytics as a Service' in the webpage of the project since its features, given below, are pretended to enable this and other use cases.

- Sahara will be designed as an OpenStack component.
- Managed through the dashboard of OpenStack or a REST API.
- Integration with other tools like Apache Ambari and Cloudera Management Console (both mentioned later in this same chapter).
- Predefined templates of Hadoop configurations with ability to modify parameters.

Sahara will communicate with other components from OpenStack for the part of provisioning, storage and authentication of the clusters of Hadoop; this gives a hint that OpenStack is somewhat ready to be used as a platform in which a DAaaS service could be implemented.

While there is little literature available about the Sahara project as its current stable version is labeled 0.3, it can be found literature about the use of OpenStack as a platform for deploying Hadoop in the Cloud manager and therefore be prepared for DAaaS in [14] and [15]. As we can see, Sahara from OpenStack aims to prepare the Cloud manager with enough potential to provide elastic Hadoop capabilities by means of other software like Apache Ambari. However, in the documentation is stated that it follows a burst scheme of operation in which the user will define a cluster size, version of Hadoop and other parameters, once this is done Sahara will be in charge of creating the cluster with the specific parameters and once it is unused it will terminate it liberating the Cloud from said cluster or do other operations in the cluster like add or remove nodes as the user demands. Taking into account this scheme it is clear that the solution intended for OpenStack is not all too elastic as it will in most cases have a fixed size that is user-defined and while it can grow by means of user intervention it is unable to do so in a more autonomic way and in this way become completely elastic.

2.1.3 Cloudera Management Console

Cloudera is the biggest company backing up Hadoop, it not only has one of the founding fathers of Hadoop but also provides services and training as well as their very own distribution of Hadoop called CDH that is still open source. CDH, as implied in their webpage [16], is a distribution of Hadoop with a few enterprise-thought additions, one of which is the Management Console.

In the documentation available in [17] and specifically in the Cloudera Manager API, there is a section talking about the automation of clusters (specific to Hadoop). In their words:

"We can install and configure Hadoop according to precise specifications using a powerful yet simple abstraction — using Cloudera Manager's open source REST API.
This is what our automation system does:

- Installs the Cloudera Manager packages on the cluster.
- Starts the Cloudera Manager server.
- Uses the API to add hosts, install CDH, and define the cluster and its services.
- Uses the API to tune heap sizes, set up HDFS HA, turn on Kerberos security and generate keytabs, customize service directories and ports, and so on. Every configuration available in Cloudera Manager is exposed in the API.

The API also gives access to management features, such as gathering logs and monitoring information, starting and stopping services, polling cluster events, and creating a disaster recovery replication schedule. It can be used to stop any service, without worrying about any additional steps. (HBase needs to be gracefully shutdown.) We use these features extensively in our automated tests. As Cloudera Manager adds support for more services, their setup flows are the same as the existing ones."

From this information it seems clear that there is a high level of automation at least for setting up the clusters and monitoring specific to their distribution of Hadoop (CDH3). Cloudera even goes as far as having a patent [18] where they explain that anyone who has setup up a cluster from scratch is well aware of how challenging it can be: every machine has to have the right packages installed and correctly configured so that they can all work together, and if something goes wrong in that process, it can be even harder to nail down the problem. Based on that problem, they claim to have a solution with the provision of the Management Console, as the examples explained before the elasticity using CDH is not as elastic as it pretends to be, as in general it will require numerous user interactions to tune the cluster for provision or de-provision the cluster in existence. Besides their Hadoop in the Cloud offering, something of note as an add-on is their backup and disaster recovery; something that PANACEA aims to also provide within its innovations, proactively. In this case Cloudera disaster recovery enables data protection in the Hadoop platform, sadly it is required a specific minimum version to work as well user configuration to actually be usable.

2.1.4 Qubole Data Service

Qubole Data Service by Qubole Inc. [19] is a Self-Service Platform for Big Data Analytics and is probably one of the top alternatives competing with PANACEA in the DAaaS Use Case. There is no official information of all the customers that Qubole have but some big names included in the public webpage include Pinterest, Oracle Data Logix, Universal Music Group, Quora and Flipboard.

The most interesting fact about Qubole is that they claim to have the first industry auto-scaling Hadoop cluster [20], which is part of what the DAaaS use case of PANACEA will do and is something that it is vital for the DAaaS use case we are working in for bringing elasticity in an autonomous and proactive manner.

While briefly mentioned in [21] as a newer entrant to the big data technology by combining the best of existing cloud-based infrastructure services with more advanced data and storage management capabilities, it is a serious contender for Hadoop in the cloud as one of its core features, auto scaling which spins up users' clusters only when a job begins, then automatically scaling or contracting clusters based on the workload, and spinning down the servers once the job is done, is something of great value in an autonomic and proactive environment such as the one PANACEA aims to provide.

There is not enough literature about Qubole available except for what it can be found in its webpage, Wikipedia and some mentions in Big Data services providers in some literature. As such we can only say that their auto-scaling features as well as its hybrid use of AWS as its cloud provider are something to lookup closely in the future.

2.1.5 HortonWorks Data Platform

HortonWorks is another big name in the Hadoop scene, it provides expertise in Hadoop ranging from training to deployment. HortonWorks Data Platform (HDP), is a specially designed Hadoop distribution, quite similar to Cloudera's CDH, which provides an enterprise ready data platform to enable organizations to harness the power of Hadoop. In their webpage [22], HortonWorks defines the features available in this product, including the fact that it leverages on YARN (Yet Another Resource Manager, part of the Hadoop framework) for the use of different technologies for data access to the HDFS part of Hadoop; it also includes various other options more related to enterprise use such as security and governance.

While it was originally not for cloud, it has teamed up with RackSpace [23] to deploy an OpenStack-based Hadoop offering for the public and private cloud and as such it is compatible with OpenStack Sahara. We found evidence of the use of HDP in the cloud in [24] and [25], but it does not really differ from other options like Cloudera or the OpenStack Sahara add-on in the way that it does not provide any elastic appeal by itself for using it, instead it relies in other technologies such as its compatibility with Apache Ambari or the mentioned OpenStack Sahara in the form of a plugin specially for it. As such, its importance for our work is minimal and will not be considered a candidate to rival our intended work.

2.1.6 Hadoop on Google Cloud Platform

This section could not be done without one of the main reasons Hadoop exists in the first place. Hadoop is available in the Google Cloud Platform as a set of setup scripts and software libraries optimized for Google infrastructure, bringing ease of use and high performance to data processing with Hadoop, [26]. With this description in mind we can assume that Google uses a heavily modified Hadoop implementation specific to the technologies that they possess, as they also mention the use of a connector for Hadoop which enables to perform MapReduce jobs directly on data in Google Cloud Storage, without copying to local disk and running Hadoop Distributed File System (HDFS). The connector simplifies Hadoop deployment, reduces cost, and provides performance comparable to HDFS, all while increasing reliability by eliminating the single point of failure of the name node.

There is little information about how elastic is this kind of deployment, Qubole mentions availability to work with the Google Cloud Platform but only as a platform to instantiate its Hadoop in the cloud, in the documentation provided in their webpage, there exists only a quick guide that teaches how to deploy a cluster for the first time, by giving some parameters similar to what OpenStacks's Sahara does. We believe we can safely assume that Hadoop in the Google Cloud Platform is not ready for elasticity as we defined at the beginning of this work.

2.2 Monitoring tools

Some of the alternatives presented in the previous section make reference to different tools used or aimed to be used to leverage the monitoring part of a Hadoop cluster including the need for metrics collection, intrinsic part of this work in the following chapters as it will be part of the tools required to know what is happening under the hood and make decisions based on it. In PANACEA we will have specific monitoring tools which may not be comparable with the following alternatives nevertheless it is important to study them and understand what are the alternatives and how they work to monitor a cluster as it can serve as a starting point for our own monitoring alternatives for the use case at hand.

2.2.1 Ganglia

Ganglia, web page available in [27], is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. Their creators assure that such system is in use to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes; also they publish some of the enterprises or institutions like Twitter, Etsy, Motorola, Harvard and even Microsoft and the U.S. Air Force there is even a demo available at http://ganglia.wikimedia.org/latest/ which shows how does Ganglia work and what are the options available for monitoring.

Ganglia, as shown in [28], [29] and [30], is a powerful tool that provides simple architecture, that provides a fast and robust implementation which leads to good scalability, robustness, and low per-node overheads; characteristics very appreciated in a monitoring tools as it needs to be as unobtrusive as possible to serve its main function, it is by no means perfect but its traits and support are sufficient to be considered as a monitoring alternative.

Hadoop is actually ready to use Ganglia for monitoring, as demonstrated in [31], as it requires little configuration within its configuration file to be able to communicate its metrics to the configured Ganglia server; this confirms that Ganglia is a tool that we can consider using for metrics collection that will have the full support of Hadoop.

2.2.2 Nagios

It is another one of the most known monitoring software for enterprises and has proved to be used for Hadoop monitoring in various cases including the one found in [31]. The first information that we can find is in their webpage, available in [32], and it is said to be marketed as the industry standard in IT Infrastructure Monitoring. Nagios is open source software licensed under the GNU GPL V2. The most interesting features available that could help monitoring a Hadoop cluster includes:

- Monitoring of network services (SMTP, POP3, HTTP, NNTP, ICMP, SNMP, FTP, SSH)
- Monitoring of host resources (processor load, disk usage, system logs) on a majority of network operating systems, including Microsoft Windows with the NSClient++ plugin or Check MK.
- Remote monitoring supported through SSH or SSL encrypted tunnels.
- A simple plugin design that allows users to easily develop their own service checks depending on needs, by using their tools of choice (shell scripts, C++, Perl, Ruby, Python, PHP, C#, etc.)
- Available data graphing plugins
- Contact notifications when service or host problems occur and get resolved (via e-mail, pager, SMS, or any user-defined method through plugin system)
- The ability to define event handlers to be run during service or host events for proactive problem resolution

The last feature is heavily used in another Nagios product denominated Nagios Reactor, using this product anyone with access to it can setup, edit, and organize automation rules and Event Chains to maintain an efficient and effective network infrastructure; it is not clear whether this product or Nagios in general can change something besides the network infrastructure. This feature of Nagios, while not specific to the case of this specific work, could be very valuable and a direct competitor to PANACEA since the networks are one of the most important parts of a Cloud infrastructure and it should be included in the proactive management aimed to be delivered by the intended framework.

It is undeniable that Nagios offer a big set of tools for monitoring the cluster not only for Hadoop but in general usage, very important for the Cloud. It is also complimented with other features including a Splunk add on as mentioned in [33] and being demonstrated how the metrics acquired in Nagios for Hadoop can be used for improving security, documented in [34].

2.2.3 Splunk App for HadoopOps

Splunk [35] is a corporation which produces software for searching, monitoring, and analyzing machine-generated big data, via a web-style interface. They have a complete offering of tools, products and services for meeting its purposes; the most prominent of it being Splunk Enterprise. Among their offering for Big Data there exists one product, known as Hunk, specifically designed for Hadoop clusters.

Since Hunk can also manage NOSQL databases, there exist various applications that are pluggable to Hunk via their own interface and can extend the "basic" functionality of it. It is here that the Splunk App for HadoopOps comes into play; this pluggable app can collect and correlate events and run-time metrics from every service on every host, every job from every user. It is also said to "provide total visibility into Hadoop's operation status, search across the entire cluster in real-time, troubleshoot and analyze Hadoop with rich, interactive views" [36]. In short, here is the list of features of such app:

* Visualize cluster resources with real-time heat map of performance metrics

* Filter, sort, and analyze jobs by user, duration, slot usage, and type

* Search and correlate events and metrics from every service, host, job, and user.

* Alert and notify stakeholders on events using automated proactive health checks

As we can see, it is a solid alternative to monitor Hadoop and perform actions manually by paying attention to the diverse alerts and notifications that the app sends to improve the QoS of a Hadoop cluster. It is true that evidence of its use was quite difficult to find, at least on scholar level since this is a very enterprise-centered solution, so we depend on some information that can be found in its webpage. It is actually briefly mentioned in a chapter in the book found in [33] and in a comparative analysis available in [37].

2.2.4 ZABBIX

ZABBIX [38] is another enterprise a little similar to Nagios, since it states that they provide "an Enterprise-class Monitoring Solution for everyone". Originally released in 2001, ZABBIX as an enterprise established in 2005 devoted to development of open source software for monitoring of networks and applications. Apart from that the company offers a wide range of professional services designed to fit every customer's unique business demands including implementation, integration, custom development and consulting services as well as various training programs.

ZABBIX is among its features, open source and offers several monitoring options:

- Simple checks can verify the availability and responsiveness of standard services such as SMTP or HTTP without installing any software on the monitored host.
- A ZABBIX agent can also be installed on UNIX and Windows hosts to monitor statistics such as CPU load, network utilization, disk space, etc.

- As an alternative to installing an agent on hosts, ZABBIX includes support for monitoring via SNMP, TCP and ICMP checks, as well as over IPMI, JMX, SSH, Telnet and using custom parameters. ZABBIX supports a variety of real-time notification mechanisms, including XMPP.

Compared to other competitors it seems to fall short but evidence of its use in the cloud can be found in [39] as well as in [40]; in one of the studies they develop a framework for virtual machines for Cloudstack platforms, this is very related to our present case as we will see in the following chapters while the other builds a scalable architecture for real-time monitoring in systems, a good asset for proactive management of cloud infrastructures, also mentioned in [41] as part of the open source tools for management of Cloud. With this information at hand we can safely say that ZABBIX is used in the Cloud for monitoring certain aspects specially for networking and as with the case of Nagios will not be considered in this work but could be helpful to be considered in the final PANACEA framework.

2.3 Automation Tools

Probably one of the most important traits of this work is looking at how to automate the Hadoop cluster for the cloud while also automating the growing and de-growing of resources autonomously to provide the aimed elasticity. In previous section we have saw that some of the alternatives for Hadoop in the Cloud provides some elasticity by means of using proprietary technologies to provide it or by leveraging part of the elasticity using technologies that can ease this process. As such, we will look in closely any option that is available commercially to provide elasticity to Hadoop in the Cloud to compare to what we aim to provide in the end.

2.3.1 Apache Ambari

Apache Ambari is one of the multiple solutions that the Apache software foundation develops to give better tools to administrators, developers and users [42]. This solution, however, is aimed at making Hadoop management simpler by developing software for provisioning, managing, and monitoring Apache Hadoop clusters. Ambari provides an intuitive, easy-to-use Hadoop management web UI backed by its RESTful APIs.

It has several interesting features for easing the administration of a Hadoop Cluster, either physical or in the Cloud and it includes:

- Provision a Hadoop Cluster: Ambari provides a step-by-step wizard for installing Hadoop services across any number of hosts and handles configuration of Hadoop services for the cluster.
- Manage a Hadoop Cluster: Ambari provides central management for starting, stopping, and reconfiguring Hadoop services across the entire cluster.
- Monitor a Hadoop Cluster: Ambari provides a dashboard for monitoring health and status of the Hadoop cluster. It will send emails when attention is needed (e.g., a node goes down, remaining disk space is low, etc.).

One of the interesting traits of Ambari is that it leverages all the monitoring part with Ganglia (studied before) for collecting metrics such as CPU utilization, I/O operations per second, average memory utilization and average network latency. It also leverages the alerting part with Nagios for providing alerts on service states and is configurable for new customized alerts.

Ambari helps the provision part of Hadoop as it requires minimum user interaction once it is fully configured; using the wizard provided, any user may be capable of creating its own cluster and add

or reduce resources as needed. It also allows the use of "blueprints", configured cluster settings that once configured can be run and automatically deploy the cluster specified in said "blueprints".
Ambari is highly regarded in documentation found in [43], [44] and [28] mainly because it not only offers simplicity for Hadoop provisioning, managing and monitoring but it also offers other ways of communication with the tools using its REST API. While this tool does not make elastic a Hadoop cluster per se, it can be used with other options to provide users with elasticity in the Cloud. The fact that this tool is available and is endorsed by Apache (which also provides Hadoop) is a testimony that Hadoop can be made elastic with some ease, the right tools and some customization.

2.3.2 SEQUENCEIQ™ PERISCOPE + BAYWATCH

SEQUENCEIQ is a small startup based in Budapest, Hungary; even though they are very new, HortonWorks has already acquired them to provide better tools to their own offering of products and services. Due to its novelty it is near to impossible to find scholar papers talking about besides what we can learn from its webpage available in [45] and their blog found in the same page.
While the options presented before the offer from SEQUENCEIQ were quite near to what is expected from PANACEA for the DAaaS use case, the tools built by SEQUENCEIQ are a much better fit since they perform specifically cases like the one PANACEA desires; although for now it seems to be limited to provide monitoring and automation to Hadoop clusters using an array of existing tools (including Apache Ambari) in combination with their own developed tools known under the name of Periscope and its monitoring companion, Baywatch both part of a much larger tool called Cloudbreak.
The purpose of Periscope is to bring QoS to a multi-tenant Hadoop cluster, while allowing applying SLA policies to individual applications. As part of Periscope they have a Hadoop cluster monitoring solution, Baywatch; this provides Periscope with the ability to monitor the application progress, the number of YARN containers/resources and their allocation, queue depths, and the number of available cluster nodes and their health (RAM available, Disk usage, networking, etc.).
For this purpose the developers at SEQUENCEIQ build their solutions using other powerful and open-source options that cover their requirements, as it is made clear by the description of its monitoring tool by using the following:

- Logstash for log/metrics enrichment, parsing and transformation
- Elasticsearch for data storage, indexing
- Kibana for data visualization

They also put to manifest that they use the metrics2 framework given by Hadoop to retrieve the metrics required to be monitored as well as using Apache Ambari to send and deploy the cluster of Hadoop itself.
A video of the solution provided by the developers running with an example that shows how to deploy cluster as well as giving a glimpse of how the SLAs are used to increase/decrease the cluster, can be seen in the following link: https://www.youtube.com/watch?v=E6bnEW76H_E

2.3.3 Apache Slider

Apache Slider [46] is an interesting project being in incubation by the Apache Software Foundation. Currently being in development the webpage describes Apache Slider as a YARN application to deploy existing distributed applications on YARN, monitor them and make them larger or smaller as desired -even while the application is running.
Applications can be stopped then started; the distribution of the deployed application across the YARN cluster is persisted —enabling a best-effort placement close to the previous locations.

Applications which remember the previous placement of data (such as HBase) can exhibit fast start-up times from this feature.
YARN itself monitors the health of "YARN containers" hosting parts of the deployed application -it notifies the Slider manager application of container failure. Slider then asks YARN for a new container, into which Slider deploys a replacement for the failed component. As a result, Slider can keep the size of managed applications consistent with the specified configuration, even in the face of failures of servers in the cluster -as well as parts of the application itself; some of the features are:

- Allows users to create on-demand applications in a YARN cluster
- Allow different users/applications to run different versions of the application.
- Allow users to configure different application instances differently
- Stop / Restart application instances as needed
- Expand / shrink application instances as needed

The tool persist the information as JSON documents in HDFS and once the cluster has been started, the cluster can be made to grow or shrink using the Slider commands based on a java command line application; the cluster can also be stopped and later restarted.
Due to its novelty, no literature could be found about it except what is available in their page and different sections. We hope that in the future when it becomes a complete release the first uses and literature appear. As for its use, it could be the case that since it aims to be a pluggable solution which is part of the YARN component of Hadoop it can be included in future versions of the DAaaS offer to provide better tools and results for both providers and users.

2.4 Other Related Technologies

In the last few sections we have looked through different commercial alternatives that are out there and could be a direct competitor to what PANACEA aims to deliver in the near future. However, the big data scene is very dynamic and more and more new technologies are born in a day-to-day basis inside the academia environment, in many regards academia is bolder to try new ways to tackle the problems of autonomous big data solutions or proactive frameworks for the Cloud. Due to this fact, we use this section to study some interesting alternatives and projects done in academia that can be related to our work in this thesis or even in PANACEA.

2.4.1 Starfish by Duke University

One of the interesting projects in academia is that of Starfish [47] by a group of investigators at Duke University, California, US. In this project they attempt to demonstrate a self-tuning system for Big Data Analytics, something that is quite similar to the DAaaS use case of this work.
In the paper, they describe the system as follow “Starfish builds on Hadoop while adapting to user needs and system workloads to provide good performance automatically, without any need for users to understand and manipulate the many tuning knobs in Hadoop”. They go about describing the different features that the users expect from a system for big data analytics and how Hadoop has the core mechanics to provide them, such features are described below:

- Magnetism: A magnetic system attracts all sources of data
- Agility: An agile system adapts in sync with rapid data evolution
- Depth: A deep system supports analytics needs that go far beyond conventional rollups and drilldowns to complex statistical and machine-learning analysis
- Data-lifecycle-awareness: A data-lifecycle-aware system goes beyond query execution to optimize the movement, storage, and processing of big data during its entire lifecycle.

- Elasticity: An elastic system adjusts its resource usage and operational costs to the workload and user requirements.
- Robustness: A robust system continues to provide service, possibly with graceful degradation, in the face of undesired events like hardware failures, software bugs, and data corruption.

However, the investigators denote that the problem with Hadoop is that it relies too much in the user or the system administrator to tune up and optimize MapReduce jobs and the overall system at various levels. The novelty of Starfish as the investigators put it is that they focuses simultaneously on different workload granularities—overall workload, workflows, and jobs (procedural and declarative)—as well as across various decision points—provisioning, optimization, scheduling, and data layout. Sadly, up until now this system only works with Hadoop 0.20.x and 1.0.3 which gives advantage to PANACEA as it aims to deliver the DAaaS use case using the more advanced and optimized Hadoop 2.x.

We strongly recommend reading the whole paper as it is quite interesting and is available in [47] or visit the webpage of the project which can be accessed via [48].

2.4.2 Data-Centric Heuristic for Hadoop Provision

Other interesting project is this heuristic approach, captured in [49], which proposes a heuristic approach to reduce the operational cost of virtual machines (VMs) running Hadoop. As the authors portray, the heuristic is simple and effective, it scales the number of Hadoop nodes based on the type and size of the job submitted. They validate such heuristic using the Hadoop word-count example on different data samples. The implementation is independent of the cloud provider; hence, the heuristic is applicable to both private and public Cloud and an interesting characteristic for them.

This heuristic technique aims to ensure the near optimal utilization of compute resources. Said heuristics works on partial input data and determines the total resource requirement to crunch complete data. In the paper they even describe that Amazon EMR is a classic example of a well-managed Hadoop provisioning but EMR is tied to the Amazon cloud and the resources are also not well optimized.

The authors do a complex calculation that can be resumed as follow, if one VM does the work in 10 hours, therefore 10 VMs will then finish the task in 1 hour. They calculate the optimum number of VMs for running Hadoop on public cloud for the data and the job by approximating the shuffle time and transfer time. The shuffle time depends on the type of network and speed. The transfer time depends upon the hardware. This is a very interesting project and something of note for the future of DAaaS either for PANACEA or any other related project as it could ease the autonomic provisioning of Hadoop in Cloud.

As it is not the intention of this work to describe the nuts and bolts of this project, we recommend reading the paper provided by the authors in [49] for more information about their approach.

2.4.3 Self-Healing and Hybrid Diagnosis

In [50], a paper from 2009, the authors already comment that Cloud computing requires a robust, scalable, and high-performance infrastructure so to provide a reliable and dependable Cloud computing platform, it is necessary to build a self-diagnosis and self-healing system against various failures or downgrades. Their solution to fulfill the self-diagnosis and self-healing requirements of efficiency, accuracy, and learning ability, a hybrid tool that takes advantages from Multivariate Decision Diagram and Naïve Bayes Classifier is proposed.

They describe Self-healing as the capability to discover, diagnose, and react to system disruptions.
The main objective of adding self-healing features to any system is to maximize the availability, survivability, maintainability, and reliability of the system. Systems designed to be self-healing are able to heal themselves at runtime in response to changing environmental or operational circumstances.
The rationale to make this possible is explained in the following paragraph; to make self-healing more tractable, they present consequence-oriented diagnosis and recovery. Even though there are numerous and various bugs, the consequence on the host's performance may be similar. Example, a memory leak may be caused by forgetting to release memory occupied by some objects when deleted. As a result, after running a long time, the host fails to properly work due to the exhaustion of its memory. Such a bug may exist at an arbitrary module/class/function/line in different programs. However, wherever the error exists or whatever types of programs they are, the consequence on the host is similar, i.e., memory consumption. Therefore, consequence-oriented healing is designed to help the hosts recover by reclaiming the leaked memory without stopping the current processes or rebooting the computer.
In the paper they describe a system using self-healing and self-diagnosis and applied it to three often occurring malfunctions in computing systems: CPU Overload, Memory Leak, and High Average Disk Queue Length to test its results, however as they mention much work is still needed to be completely viable. What is very interesting is that such an 'old' paper could come with an appealing solution for self-healing in a Cloud infrastructure; PANACEA has in its own right its specific solution for providing self-healing and we believe it will be more viable and effective than the one presented here by the authors.

2.4.4 Resilin: Elastic MapReduce

Resilin is quite recent, presented in 2013 [51], and aims to provide elasticity for MapReduce by leveraging from the Amazon Web Services (AWS). In it, the authors present the design, implementation, and evaluation of Resilin, a novel EMR API-compatible system to perform distributed MapReduce computations. Resilin goes one step beyond Amazon's proprietary EMR solution and allows users (e.g. companies, scientists) to leverage resources from one or multiple public and/or private clouds. This gives Resilin users the opportunity to perform MapReduce computations over a large number of potentially geographically distributed resources.
The author define that Like the Amazon EMR service, Resilin provides a web service acting as an abstraction layer between the users and Infrastructure-as-a-Service (IaaS) clouds as well as for interoperability reasons Resilin implements the EMR API. This allows users to reuse existing Amazon EMR tools (e.g. CLI) to interact with the system. Resilin implements most features supported by the Amazon EMR API. Users can use the EMR API to submit job flows, query job flow statuses, add new steps to an existing job flow, change the number of resources by adding or modifying groups of VMs, and terminate job flows. Moreover, unlike Amazon EMR, Resilin allows users to remove core VMs from the Hadoop cluster. In the end, once deployed, Resilin takes care of provisioning Hadoop clusters and submitting MapReduce jobs thus allowing the users to focus on writing their MapReduce applications rather than managing cloud resources. Resilin enables the execution of MapReduce jobs across geographically distributed resources with only a limited impact on the jobs execution time which is the result of intercloud network latencies. In the future we plan to extend the system by adding features allowing to automatically scale the execution platform, based on MapReduce jobs profiles and high-level objectives (e.g. meet a deadline, conserve energy).

As with other solutions mentioned in this section, we strongly recommend to read the paper as they describe the whole architecture and testing they did to validate their novel solution. We cannot really compare Resilin with the DAaaS use case as it is a solution that needs to be mounted on top of AWS to be capable of using other clouds, the DAaaS use case aims to be agnostic to the cloud infrastructure it runs in. This does not mean we can discard it, as the proposed future work of Resilin seems like interesting features to provide.

2.4.5 Automatic Optimization of MapReduce Programs

In this paper, [52], the authors set the bar high for providing techniques to automate the settings and tuning parameters for MapReduce programs. The objective is to provide good out-of-the-box performance for ad hoc MapReduce programs run on large datasets. This feature can go a long way towards improving the productivity of users who lack the skills to optimize programs themselves due to lack of familiarity with MapReduce or with the data being processed. We can honestly say that this is quite an ambitious goal.

The authors talk heavily about the different alternatives and routes that they could take to provide such a solution and in the end define the following agenda:

- Conduct a comprehensive empirical study with a representative class of MapReduce programs and different cluster configurations to understand (and potentially model) parameter impacts, interactions, and response surfaces. This study can inform the remaining steps.
- Develop a cost model that can deal with common classes of map and reduce functions by providing mechanisms to learn and plug in profiles for these functions. At the same time, it is possible that the best cost model that works at scale is a collection of parameterized rules.
- Designing an efficient sampling harness once they possess a good understanding of cost models as well as parameter impacts and interactions as a function of the properties of the job, input data, and allocated resources.
- Tune the performance of a MapReduce program that is run repeatedly (e.g., for daily report generation) and whose current performance is unsatisfactory.
- Insights from automatic optimization of single MapReduce programs will be crucial while addressing other optimization problems in the MapReduce framework.

All in all, the paper is quite interesting to read and make some good demonstrations and validations of what it aims to provide in the future. As we said before, this is quite ambitious as PANACEA is and some asseverations and comparisons of this study could be taken into account for the future.

While this the last project we include in the state of the art, we are sure that other projects have sprung in diverse settings in Academia that could benefit or compete with what PANACEA aims to bring to Cloud infrastructures, more about this will be said in the conclusion of this part.

2.5 Conclusion of State of the Art

We shall attempt to evaluate all of the different technologies we have talked about in this part of the work before continuing in to explaining what PANACEA, the DAaaS use case and the work around it was. We have seen 4 different technologies, first Hadoop in the Cloud, this was required since the foundation of the DAaaS use case is precisely this popular framework. We also commented about the alternatives in active monitoring of Cloud infrastructures, while this is not a big part of this work, it is interesting since PANACEA will require a monitoring entity of some sort to be able to deliver the aimed proactive and autonomic properties. To close this summary the automation tools

specific to Hadoop as well as some interesting projects in academia were reviewed. While the list of automation tools is quite short and only provides certain levels of automation is noteworthy that they are solutions aimed at this nowadays, however the PANACEA framework is bound to be more effective in this regard as the solutions explored in this part of the work.

As we have seen during this exercise of state of the art, most of the different solutions used and found nowadays, with the small exception of SEQUENCEIQ's and Qubole's solutions, are semi-automated solutions. They only provide a set of alarms and/or GUI to monitor and understand what is happening in the Hadoop Cluster or any other IT infrastructure; the actions that need to be performed are still done manually generally by doing some intermittent actions like sending an e-mail or a SMS so the error or problem can be corrected by the administrator or user in turn.

All in all we can conclude that as of today, no commercial or academia options available now can provide Hadoop in Cloud with elasticity that is done autonomously or proactive. Much of the elasticity that can be found for Hadoop or even other tools require that a user predefines how elastic their cluster can be and then in most cases hope that this will be enough to fulfill the required resources needed at any given time during the functioning life of said cluster.

PANACEA as a project will do more than just provide auto-scaling or Hadoop in Cloud for Cloud infrastructure, it is actually a complete set of tools and technologies that work together to provide also:

- Self-healing
- Self-configuration
- Self-optimization
- Self-protection

While the Cloud infrastructures have undoubtedly evolved and improved in big steps, there is still much work to be done towards an autonomic, better administered and failure-safe Cloud. It is due to this that PANACEA is decided to be a required solution and evolution of the Cloud infrastructure, as it will not only benefit the companies providing the service but also the day-to-day user as it will prepare the Cloud for more and better services which the average user is sure to take advantage. In the following chapter we will talk a little about how PANACEA pretends to be the spearhead towards the next evolution of the Cloud infrastructures.

CHAPTER 3 - PANACEA PROJECT

PANACEA is an interesting and extensive project currently in development by a consortium of both academia and technologic industry firms, in the following chapter we will like to give a broader picture of what PANACEA is, what it aims to do and how will it provide such objectives. We also review the use case specific to this work, we describe its parts, the innovations that are more kin to our work as well as the architecture envisioned to be delivered in the future once PANACEA is completed.

3.1 PANACEA ARCHITECTURE

PANACEA proposes innovative solutions for a proactive autonomic management of cloud resources. Advanced Machine Learning (ML) techniques are used to predict anomalies (like time to failure of applications, violation of expected response time of services or DDoS attacks) before they occur. This enables to act in advance by proactively reconfiguring the system, rather than just reacting. Expected benefits include:

(a) Higher availability, by predicting and reacting to failures before they occur,

(b) Higher security, by recognizing APT (Advanced Persistent Threat) attacks in their first stages and

(c) Higher performance, by predicting workload increases or performance bottlenecks and adapting the capacity in advance.

The PANACEA solution is targeted towards critical services having high-availability and high-reliability requirements. With PANACEA, these services will have the autonomic properties we have talked before but briefly explained here:

- *self-healing* against anomalies by recovering from multiple failures, and using proactive rejuvenation of applications and servers for preventing crashes and increasing the availability, predicting the threshold violation of response time of servers,
- *self-configuring* by efficiently mapping user's requirements onto distributed clouds and configuring on-the-fly in the presence of anomalies,
- *self-optimizing* using proactive migration of virtual machines from one cloud resource to another, maintaining the quality of service of end-to-end flows despite path outages and performance failures of the Internet,
- *self-protecting* using proactive reconfiguration of overlay networks to protect against DDoS attacks.

The proposed solution is based on the following components:

- A highly scalable monitoring system based-on interacting agents that can read computing and network sensors to collect relevant parameters. The monitoring agents can make autonomous decisions on where to focus the monitoring effort.
- Prediction models from advanced machine learning techniques allowing the prediction in advance of software and hardware failures from observations.
- A cloud management platform for autonomic services providing self-awareness and self-configuration through sensors and effectors that allow them to make and take proactive reconfiguration decisions (rejuvenation of applications, job migration…).
- Online optimization of the overlay network between clouds in order to quickly detect failures of Internet paths and to adapt the routes used to the congestion in the Internet.

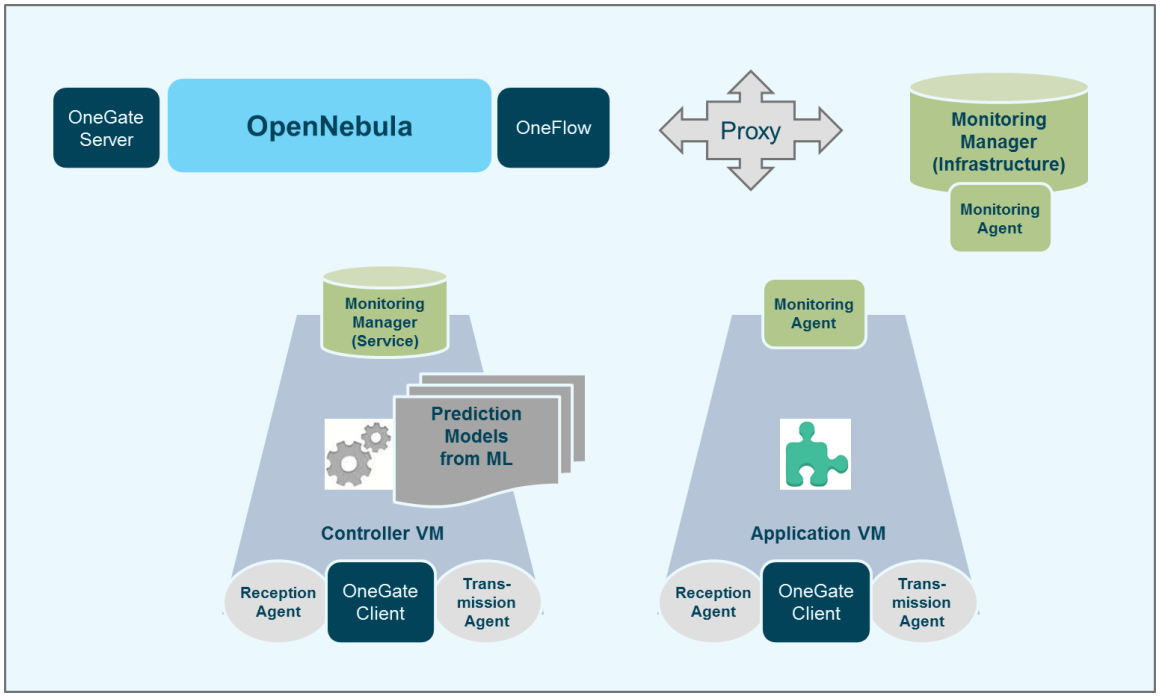

Figure 1: PANACEA Distributed Architecture showing its components

As it can be seen in the figure, OpenNebula acts as the Cloud Manager, managing both physical and virtual resources. On top of it, OneFlow acts as the Service Manager, allowing services to be deployed and managed as a whole. The OneGate Server provides self-awareness and self-configuration to autonomic services, providing a REST interface for VMs to interact with OpenNebula.

There is one Pervasive Monitoring System for the infrastructure and one for each service. The infrastructure Pervasive Monitoring System consists of a Monitoring Manager, running in a physical host (which could be the front-end host running OpenNebula) or even in a VM, and one Monitoring Agent running on each physical host acting as worker node for OpenNebula.

For the Routing Overlay, the Overlay Proxy is deployed in a physical host or even in a VM.

The ML framework (Predictions Models) will allow predicting the failure time of software, or user applications running on Virtual Machines and the violation of expected response time of Cloud services. The complexity of prediction models will be reduced by removing a number of irrelevant parameters from a training data set while preserving the accuracy of predictions.

To deal with the vast number of possible resources to monitor, our main approach will consider the use of mobile agents, which will move on the Cloud, interacting with other agents, reading computing and network sensors, and making autonomous decisions on what to measure, when to report and to whom.

To ensure a reliable operation of PANACEA, replication services, leader election and distributed locking mechanisms shall be used wherever critical information is maintained.

3.2 DATA ANALYTICS AS A SERVICE (DAaaS) USE CASE

A DAaaS platform may well be a combination of IaaS and PaaS since there is an infrastructure (the Hadoop clusters and storage) as well as the Analytics part which is specific software used for analyzing the results acquired by the infrastructure. In order to validate the capability to proactively manage Cloud resources in a dynamic environment developed in PANACEA, a proof of concept was developed by Atos using technologies such as Hadoop, both for data storage using HDFS as well as

MapReduce in conjunction with algorithms in R (The R project for Statistical Computing) for providing the applications for data analytics as well as using other tools from the Hadoop framework such as Pig, Hive and Mahout, this is known inside of the PANACEA Framework as the Use Case 2: Data Analytics as a Service (DAaaS). In said proof of concept, data from the sensors of a factory were used (data provided included temperature, vibrational speed, rotatory speed, etc.). Such data will then be loaded in the DAaaS platform and used for analysis jobs to analyze their feasibility as a validation vehicle for the scientific outputs of the project.

In this regard, the DAaaS use case of this work is composed mainly of a Hadoop 2.x cluster conformed of 5 different machines (2 masters, 3 slaves) which can then scale up or down as required to provide the elasticity we desire as part of this work, in the following figure we can observe how our Hadoop cluster will look from the point of view of a user waiting to use it, this of course is just the initial setup, in the following sections we will go in deep through how the architecture of the DAaaS use case will be developed.

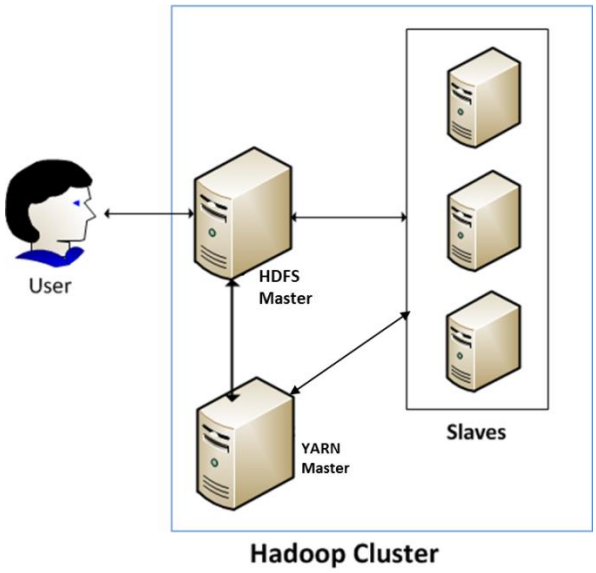

Figure 2: The initial Hadoop Cluster of the DAaaS use case.

As we mentioned before, this proof of concept is used for validating the innovations intended to be delivered by PANACEA for Cloud applications. Autonomic Cloud Service Management is thus the innovation object of this work and can be divided in two big tasks presented below:

- Autonomic management of Cloud services
- Ubiquitous monitoring of the Cloud.

In the following sections this tasks will not only be explained in what they are expected to do but also on what was the approach taken and the work done to realize the objectives of such tasks.

3.2.1 Autonomic management of Cloud services

Autonomic systems have the following properties:

- Self-configuring: They have the ability to define themselves “on-the-fly”.
- Self-healing: They discover, diagnose, and react to disruptions.
- Self-optimizing (self-adapting): They can monitor and tune resources automatically.
- Self-protecting: They anticipate, detect, identify and protect themselves against threats from anywhere.

The following is a more complete list of features that should be exposed by any Cloud manager to deploy autonomic self-managing systems:

- Self-configuring features:
    - systems can change configuration parameters (capacity, placement, connectivity...) at runtime
    - systems can connect new devices at runtime (hot plugging)
- Self-healing features:
    - systems can detect faults and recover from them
    - systems can perform software rejuvenation
    - systems can manage spares for application live migration
- Self-optimizing (self-adapting) features:
    - systems can get information of the execution environment at runtime (self-awareness) and adapt to it (self-configuration)
    - systems can manage elasticity
- Self-protecting features:
    - systems can define and manage user access
    - systems can detect attacks and recover from them
    - systems can perform backup and recovery

On the other hand, proactivity or proactive behavior refers to anticipatory, change-oriented and self-initiated behavior, which involves acting in advance of a future situation, rather than just reacting. In particular, proactive systems predict anomalies (like time to failure of Cloud applications or DDoS attacks) before they occur.

In principle, no special functionality has to be included in the Cloud manager to support proactive systems, provided that autonomic systems are supported, since the proactive behavior would rely on the systems themselves. Proactive systems provide a number of benefits, including:

- Higher availability, by predicting and reacting to failures before they occur.
- Higher security, by recognizing APT (Advanced Persistent Threat) attacks in their first stages.
- Higher performance, by predicting workload increases or performance bottlenecks and adapting the capacity on advance.

3.2.2 Ubiquitous monitoring of the Cloud

Internally known as Pervasive Cloud monitoring, this innovation is vital for obtaining a timely and accurate representation of the Cloud in order to enable proactive and autonomic service management, which depend on knowing the state of the Cloud at different granularities and levels in order to make quick and informed decisions, it is basically the innovation in charge of the decision-making for the other innovations. The solution of PANACEA provides improved monitoring of Cloud infrastructures and services using autonomic principles. It supports proactive and autonomic Cloud management by collecting multiple fine-grained metrics from the Cloud hosts, VMs, services and applications, transferring the monitoring data in a timely manner without incurring high network overhead, and making the data available to the Cloud and service managers. The proposed monitoring solution *self-optimizes by adaptively prioritizing which nodes to monitor in the Cloud*, thereby achieving low overhead while providing timely delivery of fine-grained monitoring data, and enabling an accurate representation of the state of the Cloud, which is critical for good management and allocation decisions. Keeping the overhead low while providing good accuracy is primarily achieved by using machine learning techniques based on random neural networks that continually learn which parts of the Cloud are changing the most, and focusing the monitoring on those areas as needed.

Low overhead monitoring is especially important in large Cloud infrastructures, where a high number of physical hosts, VMs and services need to be monitored concurrently. Fine-grained monitoring becomes very difficult in such large-scale deployments due to the amount of monitoring

data that needs to be collected and transferred. PANACEA's pervasive (ubiquitous) Cloud monitoring solution addresses this issue with a scalable monitoring system design where monitoring agents autonomously decide on where to focus the monitoring effort, and enables us to provide more responsive and optimal proactive and autonomic management services in large-scale Clouds.

As of writing and execution of the DAaaS use case tests this innovation has not been fully adopted in the project since it is still in development by one of the partners in the PANACEA project and thus will be added in the following phase of PANACEA development. Although we are not able to use this specific innovation, we used a simple but functional approach to have monitoring capability in our testing environment of the DAaaS use case, and we are confident that the information gathered using this approach will serve as a starting point for the monitoring innovation algorithm that will be used further along the development of PANACEA.

3.3 DAaaS ARCHITECTURE

Running big data applications on Cloud infrastructures is becoming common practice since such applications typically require a large number of worker nodes to store and process the data, and the Cloud enables easier management and better scalability and elasticity of big data applications. In this new application domain, the Cloud service of *data analytics as a service (DAaaS)* is presented, which provides an end-to-end data analytics platform for data scientists and business that captures functions of the data analytics process from data acquisition to end-user visualization, reporting and interaction.

DAaaS is, in short, an extensible analytical platform provided using a Cloud-based delivery model, where various tools for data-analytics are available and can be configured by the user to efficiently process and analyze huge quantities of heterogeneous data. An overview of the DAaaS platform is given in Figure 3, where data is captured by and stored on the platform via data suppliers, and data consumers configure and execute data analytics, visualization, and reporting jobs on the platform in order to get back results and insights.

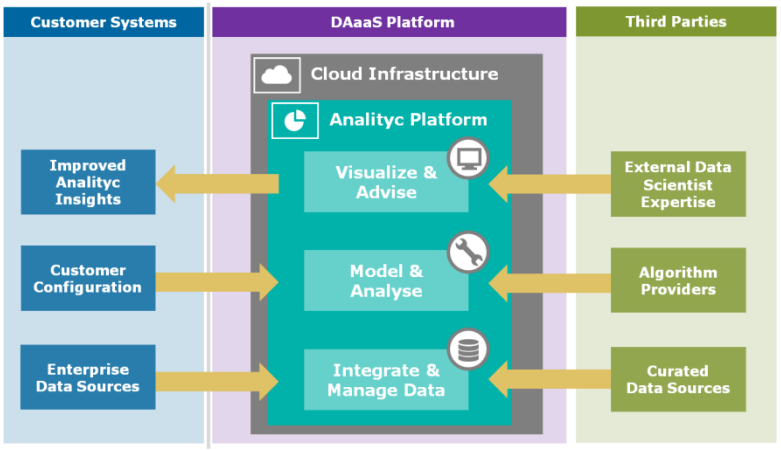

Figure 3: Diagram of the DAaaS platform

These results are generated by Analytic Apps provided as part of the platform which can be configured and extended by the end-users according to their individual needs and workflows. These workflows are built using an extensible collection of services that implement analytical algorithms, many of them based on Machine Learning techniques.

There are several end-users from diverse sectors and as such the final DAaaS platform intended to be used by ATOS using the expertise and technologies of PANACEA will be complex and rely on

various technologies for the different stages of the big data analysis chain, as show in the figure below.

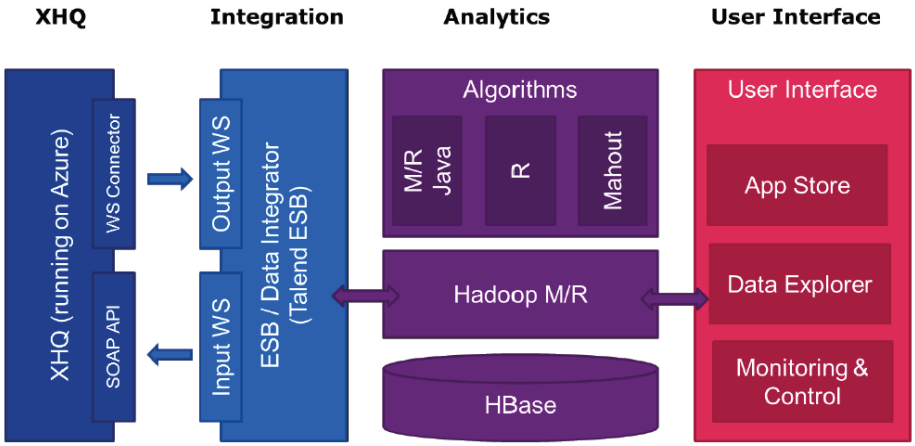

Figure 4: Architecture and technologies used in the proposed solution

The Siemens XHQ is an external service to the DAaaS solution. The integration layer just retrieves through web-services the data coming from the XHQ and feeds them to the analytics modules or stores them into the HDFS /HBase database.

The data analytics is based on Apache Hadoop; it can use Map – Reduce algorithms written in Java, Basic Hadoop tools like Pig or Hive, Apache Mahout Algorithms. It also offers integration with R statistic language. All the data coming from the XHQ service (if it is necessary to store) and the results coming from the data analytics algorithms are stored into an Apache HBase - a distributed, scalable, big datastore.

Finally, the User Interface contains the following modules: the App Store enabling users to configure the different analytics algorithms to be used with their data; the Data Explorer module to see the results coming from the different analysis; And the Monitoring and Control module.

DAaaS represents an ideal application to be executed in a Cloud environment. Its components can elastically grow in a Cloud environment to adapt to the incoming demand from the users, using as many resources as necessary at a given time. The system has to be highly available and should run without any problem as much as possible, as any downtime can have an economic impact on the service provider.

Due to time and budget constraints an alternative to this platform was made to perform a diverse range of tests that once finished and reviewed will help deploy the intended final platform. The following couple chapters devote themselves to describing the work performed to create the DAaaS algorithms as well as the different elements of this platform and tests used to evaluate its performance and usability for further usage.

CHAPTER 4 - TESTS AND EXPERIMENTS

Is the purpose of this chapter to show all of the factors used for testing the innovations to showcase the use case at hand; thus we include a brief explanation of the architecture required for doing such tests in order to be able to be replicated by anyone that have access to the tools presented here.

4.1 TESTING ARCHITECTURE

As we have said through this work, the final objective is to be able to deliver elasticity to Hadoop in Cloud however this elasticity should be done autonomously and in some scenarios, proactive. To be able to do that we first need to understand how it is done manually, so then we can teach our algorithm all the necessary steps to take into account not only to provide the elasticity we are looking for but also the problems that it could face and should be able to deal with. Doing this kind of approach on a production environment is not only risky but also not cost effective, so instead we need to devise a Cloud architecture that we can control and in which we are able to send our testing algorithms and debug whatever problem we may encounter with minimum repercussions.

In the figure below we can clearly see what the components of the architecture delivered for this use case are. Hadoop in its version 2.x requires two different master components and an infinite number of slaves or workers (also called nodes in the framework) the master components are HDFS and YARN; while HDFS is in charge of all the storage for the cluster, YARN is in charge of the correct execution of the MapReduce applications that go to this specific cluster. So as commented before the cluster for Hadoop will consist of 2 masters (the HDFS and the YARN) which consist of the NameNode (VM2) and ResourceManager (VM1) daemons, respectively. In the beginning we will have 3 slaves or Hadoop workers (default number for HDFS), each of these workers have two daemons, the DataNode (corresponding to HDFS) and the NodeManager (corresponding to YARN); this configuration is true for any Hadoop worker we add to the cluster to make it elastic. Besides the daemons, and once the cluster has received one job, the YARN will designate an AppMaster to monitor and execute said job and designate a set of resources in the form of containers that can come from different workers, the AppMaster will coordinate everything and once the job is finished return the result to the ResourceManager machine which in turns will return it to the user or machine where the job was sent from. As a final part of the architecture, we will have a Virtual Machine that will serve as the CollectD hub (for metrics collection across the cluster) and provide the PANACEA logic for autonomic and proactive management of this cluster (VM3), to be able to collect the different metrics available across the cluster, CollectD daemons will be included inside all of the different virtual machines to collect said metrics and understand what is happening in the different parts of the cluster.

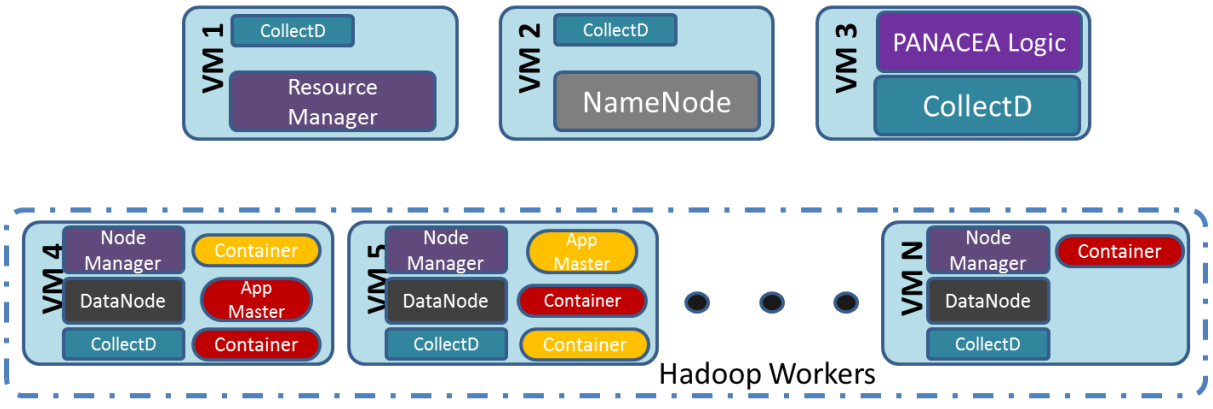

Figure 5: Architecture for testing DAaaS use case

In this regard the architecture used for testing, while a little simple, is formed by a single hardware (with different virtualized machines inside of it) and diverse open source software, in part described in the last chapters, set up so they show the different capabilities of the innovations of PANACEA for DAaaS.
After we have understood what our architecture will be we can then proceed to list both the hardware and the software used to do the testing as well as the building process we had to face to deliver the intended architecture. At the end of this chapter we will also dedicate ourselves to introduce the logic and description of the different sets of test set to be executed in the aforementioned architecture.

4.1.1 Testing architecture implementation

In the following two sections we will describe the hardware and software, respectively, which we used for the testing environment.
The hardware described and used on this environment was the minimal required to setup a test bed for the DAaaS use case and it also was specifically selected for this task in order to be totally isolated from other tasks that could outperform the tests performed.
As for the software part, most of it obeys a simple need for simplicity but at the same time of functionality towards our use case. Especially is the use of OpenNebula, [54], since some of the features intended by OpenNebula are already being provided in a non-autonomic way; configuration, optimization, protection and fault tolerance are all there but providing them in an autonomic (self-*) and proactive way, through PANACEA, will improve the availability and QoS of services. The usage of OpenNebula, is thus considered a horizontal use case inside the PANACEA project.

4.1.2 Hardware

- Server with 24 cores (with hyper-threading, only 12 physical) and 32 GB RAM, with 3TB of HD. This server runs with an installation of Ubuntu Server with SSH and the KVM hypervisor configured.

- Virtual Machine with Debian OS used for the OpenNebula Frontend, totally outside of the server but still able to communicate with it through network.

- Virtual Machine inside the KVM Hypervisor, with Ubuntu Server, to be used as the Hadoop NameNode.

- Virtual Machine inside the KVM Hypervisor, with Ubuntu Server, to be used as the Hadoop Resource Manager

- Any number of virtual machines inside the KVM Hypervisor, again with Ubuntu Server, to perform as the Hadoop Slaves (DataNodes and NodeManagers).

4.1.3 Software

- OpenNebula on its latest version as of this writing, 4.12. Easily acquired on their webpage http://opennebula.org/software/ and installed following the instructions in the documentation.

- Hadoop latest release, 2.6.0 as of writing. Acquired via https://hadoop.apache.org/releases.html

- OneGate for PANACEA, an extension for OpenNebula acquired through https://github.com/dsa-research/onegate-panacea/wiki/Installation. Bear in mind that OpenNebula already has OneGate

though it is not loaded with the PANACEA innovations but once they are tested they will be included in an OpenNebula release.

- CollectD, version 5.5, is a daemon which collects system performance statistics periodically and provides mechanisms to store the values in a variety of ways. CollectD gathers statistics about the system it is running on and stores this information. Those statistics can then be used to find current performance bottlenecks (i.e. performance analysis) and predict future system load (i.e. capacity planning).

- All the software required to make Hadoop work, which basically includes JAVA JDK (7u75), SSH as well as SSHD and RSYNC.

The final working architecture for testing is represented in the figure below. In it we can see how the communication takes place. The OpenNebula frontend will monitor the server with the hypervisor which in turn is considered as a host for OpenNebula, it will also send requests to instantiate any of the images for virtual machines to work as a Hadoop cluster, and this of course will also be monitored by the front end. Finally the VMs will use the innovations provided by OneGate to communicate with the frontend by sending their statistics and metrics required for actions to be performed in case of imminent failure or any other situations that may appear like a need for scaling the cluster.

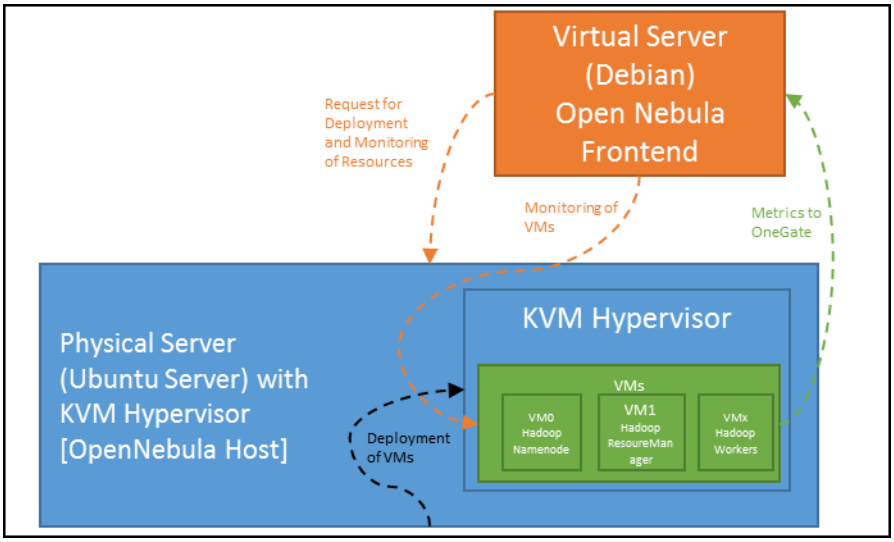

Figure 6: Final testing architecture

4.2 TESTING SETUP

The hardware and the software listed and described above cannot do much all by themselves if not setup and configured correctly, in this section we will describe the different steps performed to build the architecture presented so results acquired can be replicated if required.

4.2.1 Setting up OpenNebula

One of the first things to do is setup OpenNebula, to do so we followed the documentation that can be found in http://docs.opennebula.org/4.12/design_and_installation/building_your_Cloud/ignc.html but with a few twists.

One thing to bear in mind is that we should add the ssh keys of themselves both to the host and the front-end of OpenNebula (passwordless); this is especially true if the storage configuration obeys an ssh approach as this setup. To do so one can use any of the options below and check it by connecting through ssh to each other and to themselves to verify:

```
--- Adding the key manually to authorized_keys and known_hosts in the frontend and hosts ---
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
--- Copying the ssh by using ssh-copy-id option ---
$ ssh-copy-id -i ~/.ssh/id_rsa.pub remote-host
```

Another thing to consider for this setup is the storage configuration (step 9 of the guide of OpenNebula); it is said in this step that two types of storage need to be created, the system datastore and the filesystem datastore. If the guide is followed, however, they are created automatically. While the system datastore was not modified, we did create a new filesystem datastore to be used with the ssh driver; the template is down here and it was created using the 'onedatastore create *name_of_file*' CLI command. The output of this command should be an ID of the datastore created and it will be helpful later on when we create the images for deploying and instantiating VMs in OpenNebula.

```
NAME = test_fs
TM_MAD = ssh
DS_MAD = fs
```

Once finished with the guide, the front end and the host (our server with 24 cores in this case) will have the oneadmin account and should be able to see each other (ssh to each other) as well as being prepared to be used as a private Cloud. We will continue with the deployment of a Hadoop installation outside OpenNebula and return to it once finished.

4.2.2 Creating Ubuntu-Server image for Hadoop:

First things first, check and verify that virsh (KVM) and virt-viewer are correctly installed and being able to use; once done also verify that and iso of the Ubuntu server is available for installing it.
Once done, first create an image file for installing Ubuntu Server (v. 14.04 64bit)

```
qemu-img create –f raw /home/atos/img_hadoop/Hadoop_original.img 15G
```

Then create a domain creation xml for the server. This can be used but can be modified.

```
<domain type='kvm'>
  <name>Nebulatest</name>
  <uuid>887ee873-485a-4204-a5fd-90aac6571d09</uuid>
  <memory unit='KiB'>2097152</memory>
  <currentMemory unit='KiB'>2097152</currentMemory>
  <vcpu placement='static'>2</vcpu>
  <resource>
    <partition>/machine</partition>
  </resource>
  <os>
    <type arch='x86_64' machine='pc-i440fx-trusty'>hvm</type>
    <boot dev='hd'/>
    <boot dev='cdrom'/>
  </os>
  <features>
    <acpi/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
```

```
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw'/>
      <source file='/home/atos/img_hadoop/Hadoop_original.img'/>
      <target dev='vda' bus='virtio'/>
      <alias name='virtio-disk0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04'
function='0x0'/>
    </disk>
    <disk type='file' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <source file='/home/atos/isos/ubuntu-14.04.2-server-
amd64.iso'/>
      <target dev='hdc' bus='ide'/>
      <readonly/>
      <alias name='ide0-1-0'/>
      <address type='drive' controller='0' bus='1' target='0'
unit='0'/>
    </disk>
    <controller type='ide' index='0'>
      <alias name='ide0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x01'
function='0x1'/>
    </controller>
    <controller type='usb' index='0'>
      <alias name='usb0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x01'
function='0x2'/>
    </controller>
    <controller type='pci' index='0' model='pci-root'>
      <alias name='pci.0'/>
    </controller>
    <serial type='pty'>
      <source path='/dev/pts/1'/>
      <target port='0'/>
      <alias name='serial0'/>
    </serial>
    <console type='pty' tty='/dev/pts/1'>
      <source path='/dev/pts/1'/>
      <target type='serial' port='0'/>
      <alias name='serial0'/>
    </console>
    <input type='mouse' bus='ps2'/>
    <input type='keyboard' bus='ps2'/>
    <graphics type='vnc' port='5900' autoport='yes'
listen='172.18.10.8'>
      <listen type='address' address='172.18.10.8'/>
    </graphics>
    <video>
      <model type='cirrus' vram='9216' heads='1'/>
      <alias name='video0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02'
function='0x0'/>
    </video>
    <memballoon model='virtio'>
      <alias name='balloon0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0'/>
    </memballoon>
    <interface type='network'>
```

```
      <source network='default'/>
    </interface>
  </devices>
  <seclabel type='dynamic' model='apparmor' relabel='yes'>
    <label>libvirt-887ee873-485a-4204-a5fd-90aac6571d09</label>
    <imagelabel>libvirt-887ee873-485a-4204-a5fd-
90aac6571d09</imagelabel>
  </seclabel>
</domain>
```

Once saved with whatever name desired, use:

```
virsh create name_of_file.xml
```

And we should get an output with the ID of the VM, use this with virt-viewer to access to the VM through VNC and follow the instructions to install the Ubuntu Server. There is no special requirement or configuration to do just make sure to install the OpenSSH package. In our particular case, the username is 'hadoop' as well as the hostname of the machine. Once it is completely installed enter with your username and password and follow the next phase.

4.2.3 Installing Java, Hadoop and other configurations in the Ubuntu server.

From this image download the java sdk, 7u75[1] 64bits, from the Oracle web page and proceed to install it however one see fit but here are a few easy steps to do so.

1. Create a directory where to save java binaries: `sudo mkdir -p /usr/local/java`
2. Untar the .tar.gz file downloaded from Oracle web page on the created folder: `sudo tar xvzf jdk-7u75-linux-x64.gz /usr/local/java/`
3. We should have now a directory called jdk1.7.0_75
4. Providing this, edit the bashrc file: `sudo nano .bashrc`
5. At the end of such file add the following two lines and after doing so, save it:
   ```
   export JAVA_HOME=/usr/local/java/jdk1.7.0_75
   export PATH=$PATH:$JAVA_HOME/bin
   ```
6. Update the alternatives of the system so it can now where to get the java, javac and java ws binaries:
   ```
   sudo update-alternatives --install "/usr/bin/java" "java"
   "/usr/local/java/jdk1.7.0_75/bin/java" 1
   sudo update-alternatives --install "/usr/bin/javac" "javac"
   "/usr/local/java/jdk1.7.0_75/bin/javac" 1
   sudo update-alternatives --install "/usr/bin/java" "javaws"
   "/usr/local/java/jdk1.7.0_75/bin/javaws" 1
   ```
7. And what the defaults are:
   ```
   sudo update-alternatives --set java
   /usr/local/java/jdk1.7.0_75/bin/java
   sudo update-alternatives --set javac
   /usr/local/java/jdk1.7.0_75/bin/javac
   sudo update-alternatives --set javaws
   /usr/local/java/jdk1.7.0_75/bin/javaws
   ```
8. Finally reboot the machine and test if the java is correctly configured by executing: `$ java -version`

Now let us continue with the installation of Hadoop, download the latest version, 2.6.0. Follow these steps for the first part; Hadoop recommends that a specific group and specific user for Hadoop is created, but since this images will be used for specific Hadoop purposes we will skip the creation of such user and group.

1. Create a directory for the binaries and everything else: `sudo mkdir /usr/local/hadoop`

[1] In general terms, one may be able to use openjdk-7 but Hadoop seems to rely more heavily on the version provided by Oracle

2. Untar the Hadoop downloaded from Hadoop repository to the created folder: `sudo tar xvzf hadoop-2.6.0.tar.gz /usr/local/hadoop`
3. Change the ownership of the directory: `sudo chown hadoop:hadoop /usr/local/hadoop`
4. Create the ssh key and add it to the authorized_keys:

```
:$ ssh-keygen -t rsa -P ""
:$ ssh-copy-id -i ~/.ssh/id_rsa.pub localhost
```

5. Check that you are able to ssh to yourself before continuing by using any of the two options below:

```
:$ ssh localhost
:$ ssh ip_of_the_machine
```

6. Edit again the .bashrc file and add the following at the end of the java lines for being able to perform Hadoop commands from wherever is needed:

```
export HADOOP_HOME=/usr/local/hadoop/hadoop-2.6.0
export
PATH=${PATH}:${JAVA_HOME}/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sb
in
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
```

7. A log can be configured for both HDFS and YARN so we create one and change the owner to hadoop:

```
:$ sudo mkdir /var/log/hadoop
:$ sudo chown hadoop:hadoop /var/log/hadoop
```

8. We need to create two folders for HDFS, one for NameNode and another for DataNodes (slaves)

```
:$ mkdir –p
/usr/local/hadoop/hadoop2.6.0/hadoop_data/hdfs/namenode
:$ mkdir –p
/usr/local/hadoop/hadoop2.6.0/hadoop_data/hdfs/datanode
```

9. We will now configure all the different files needed to run Hadoop in cluster mode. Head to the hadoop folder, all of the files are under **/usr/local/hadoop/hadoop2.6.0/etc/hadoop**. The terminology used for the hostnames of the different machines that will be used to configure this files is as follow:

   Hop-NameNode (The NameNode for Hadoop HDFS component)

   Hop-ResourceMan (The ResourceManager for Hadoop YARN component)

   Hop-Worker-# (The slaves, replace '#' with the corresponding number)

   Once this terminology has been explained, let us modify the following files.

   - hadoop-env.sh

```
export JAVA_HOME=/usr/local/java/jdk1.7.0_75/

export HADOOP_LOG_DIR=/var/log/hadoop/
```

   - yarn-env.sh

```
export YARN_LOG_DIR=/var/log/hadoop/
```

   - core-site.xml

```
<configuration>
        <property>
           <name>fs.defaultFS</name>
           <value>hdfs://Hop-NameNode:9000</value>
```

```
        </property>
</configuration>
```

-hdfs-site.xml (by default hdfs replication requires 3 slaves, however you can add another property with name dfs.replication and change the value to 1, 2 or more)

```
<configuration>
        <property>
                <name>dfs.namenode.name.dir</name>
                <value>file:/usr/local/hadoop/hadoop-
        2.6.0/hadoop_data/hdfs/namenode</value>
        </property>
        <property>
                <name>dfs.datanode.name.dir</name>
                <value>file:/usr/local/hadoop/hadoop-
        2.6.0/hadoop_data/hdfs/datanode</value>
        </property>
</configuration>
```

- yarn-site.xml

```
<configuration>
<!-- Site specific YARN configuration properties -->
    <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
    </property>
    <property>
      <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
      <value>org.apache.hadoop.mapred.SuffleHandler</value>
    </property>
<!-- Should work for a resource manager. This are the addresses for the different components of YARN -->
    <property>
      <name>yarn.resourcemanager.resource-tracker.address</name>
      <value>Hop-ResourceMan:8025</value>
    </property>
    <property>
      <name>yarn.resourcemanager.scheduler.address</name>
      <value>Hop-ResourceMan:8030</value>
    </property>
    <property>
      <name>yarn.resourcemanager.address</name>
      <value>Hop-ResourceMan:8050</value>
    </property>
    <property>
      <name>yarn.resourcemanager.admin.address</name>
      <value>Hop-ResourceMan:8033</value>
    </property>
    <property>
      <name>yarn.resourcemanager.webapp.address</name>
      <value>0.0.0.0:8088</value>
    </property>
<!-- For excluding nodes (eg. scaling down)-->
            <property>
```

```
                <name>yarn.resourcemanager.nodes.exclude-path</name>
                <value>/usr/local/hadoop/hadoop-2.6.0/etc/hadoop/yarn.exclude</value>
              </property>
<!-- Next part is used for configuring the fair scheduler -->
              <property>
                <name>yarn.resourcemanager.scheduler.class</name>
                <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.F$
              </property>
</configuration>
```

- mapred-site.xml (copy mapred-site.xml.template to mapred-site.xml)

```
<configuration>
        <property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
        </property>
</configuration>
```

- slaves (since by default we need 3 slaves, we will put the 3 hostnames of such slaves here)

```
Hop-Worker-1
Hop-Worker-2
Hop-Worker-3
```

10. To finish, we need to install the context package of OpenNebula for this machine to be fully compatible with the OpenNebula software. Let us download the package and install it.

```
:$ wget 'https://github.com/OpenNebula/addon-context-
linux/releases/tag/v4.10.0/one-context_4.10.0.deb'
:$ sudo dpkg -i one-context_4.10.0.deb
```

11. Everything for running this test cluster of 5 machines is ready, so poweroff this machine and continue.

4.2.4 Monitoring tools

While everything is ready for deployment, it is required that we gather metrics for validation so we can understand what is happening when the cluster is fully functional and what do we have to pay attention in order to guarantee that we provide insights of what was and what is happening in the cluster so the innovations in charge of taking care of the different parts of the cluster work correctly and do their designed job. For this purpose we have setup Collectd as our metrics gatherer and using the metrics available from the YARN components of Hadoop 2.6.0; this will be part of the ubiquitous monitoring part of the use case.

First things first, we need to install Collectd in our virtual disk and then we need to create a set of bash scripts that will gather all the information from the components. Once we have done that, we can configure Collectd to use this bash scripts and send them to a repository that we can later access and query the results of the metrics.

To install collectd is as simply as executing the following:

```
:$ apt-get update
:$ apt-get install collectd
```

If by any reason the virtual machine does not have internet connection, there are .tar.bz2 files available at their homepage in [55] that you can download from an internet-able computer and transferred to the virtual machine to install.

Once it has been installed, we can write the required bash scripts, since there are a lot of them; they can be acquired by going to the Annex 2. Here is just an example of one of them (hadoop-yarn-rpc.sh) which basically creates a file that will gather the metrics that will be read by Collectd and fills it with the metrics each and every 10 secs. The rest of bash scripts used for monitoring can be found in the Annex 2 have the same structure but they write different values about the information provided by the components.

```
#!/bin/bash

HOSTNAME="Hop-ResourceMan"
INTERVAL=10
HADOOP_METRICS_FILE="/tmp/resourcemanager-metrics.out"
NODE_NAME="RPC-8050"

while sleep "$INTERVAL"
do

  LINE=`tail -n 100 $HADOOP_METRICS_FILE | grep "rpc.rpc: port=8050" | tail -n 1`
  TIME=`echo $LINE | cut -d" " -f 1`
  ReceivedBytes=`echo $LINE | awk '{ print $6 }' | cut -d"=" -f 2 | cut -d"," -f 1`
  SentBytes=`echo $LINE | awk '{ print $7 }' | cut -d"=" -f 2 | cut -d"," -f 1`
  RpcQueueTimeNumOps=`echo $LINE | awk '{ print $8 }' | cut -d"=" -f 2 | cut -d"," -f 1`
  RpcQueueTimeAvgTime=`echo $LINE | awk '{ print $9}' | cut -d"=" -f 2 | cut -d"," -f 1`
  RpcProcessingTimeNumOps=`echo $LINE | awk '{ print $10}' | cut -d"=" -f 2 | cut -d"," -f 1`
  RpcProcessingTimeAvgTime=`echo $LINE | awk '{ print $11}' | cut -d"=" -f 2 | cut -d"," -f 1`
  RpcAuthenticationFailures=`echo $LINE | awk '{ print $12}' | cut -d"=" -f 2 | cut -d"," -f 1`
  RpcAuthenticationSsuccesses=`echo $LINE | awk '{ print $13}' | cut -d"=" -f 2 | cut -d"," -f 1`
  RpcAuthorizationFailures=`echo $LINE | awk '{ print $14}' | cut -d"=" -f 2 | cut -d"," -f 1`
  RpcAuthorizationSuccesses=`echo $LINE | awk '{ print $15}' | cut -d"=" -f 2 | cut -d"," -f 1`
  NumOpenConnections=`echo $LINE | awk '{ print $16}' | cut -d"=" -f 2 | cut -d"," -f 1`
  CallQueueLength=`echo $LINE | awk '{ print $17}' | cut -d"=" -f 2 | cut -d"," -f 1`

  echo "PUTVAL $HOSTNAME/$NODE_NAME-ReceivedBytes/memory interval=$INTERVAL
$TIME:$ReceivedBytes"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-SentBytes/memory interval=$INTERVAL
$TIME:$SentBytes"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcQueueTimeNumOps/memory interval=$INTERVAL
$TIME:$RpcQueueTimeNumOps"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcQueueTimeAvgTime/memory interval=$INTERVAL
$TIME:$RpcQueueTimeAvgTime"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcProcessingTimeNumOps/memory
interval=$INTERVAL $TIME:$RpcProcessingTimeNumOps"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcProcessingTimeAvgTime/memory
interval=$INTERVAL $TIME:$RpcProcessingTimeAvgTime"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcAuthenticationFailures/memory
interval=$INTERVAL $TIME:$RpcAuthenticationFailures"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcAuthenticationSuccesses/memory
interval=$INTERVAL $TIME:$RpcAuthenticationSuccesses"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcAuthorizationFailures/memory
interval=$INTERVAL $TIME:$RpcAuthorizationFailures"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcAuthorizationSuccesses/memory
interval=$INTERVAL $TIME:$RpcAuthorizationSuccesses"
```

```
  echo "PUTVAL $HOSTNAME/$NODE_NAME-NumOpenConnections/memory interval=$INTERVAL $TIME:$NumOpenConnections"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-CallQueueLength/memory interval=$INTERVAL $TIME:$CallQueueLength"

done
```

There are a few more steps required to finish the monitoring of the Hadoop components. First, we need to modify the file intended by Hadoop for the metrics, this file is located in the following address: */usr/local/hadoop/hadoop-2.6.0/etc/hadoop/hadoop-metrics2.properties* and so we need to uncomment and change the following lines as so:

```
resourcemanager.sink.file.filename=/tmp/resourcemanager-metrics.out

nodemanager.sink.file.filename=/tmp/nodemanager-metrics.out

mrappmaster.sink.file.filename=/tmp/mrappmaster-metrics.out

jobhistoryserver.sink.file.filename=/tmp/jobhistoryserver-metrics.out
```

The image is almost ready, but now we need to perform some steps for Collectd to be able to collect all of the different metrics we have designated. As we said at the beginning of this chapter, we need to include a VM that will serve as the Collectd hub for metric collection, in this particular case we have designated our physical server to serve as our hub. The necessary steps to configure said hub are listed below:

1. Install collectd: Just as we did before.
2. Configure graphs: We used the following tool; https://github.com/pommi/CGP and followed the instructions available in its webpage.
3. Install Apache 2: First by sending *apt-get install apache2* and after it we send *apt-get install libapache2-mod-php5*
4. After the correct installation of apache we need to create a directory where to install the tool for the graphics, so we execute *mkdir /var/www/html/metrics*
5. We change directory to the one we just created and we then execute *git clone https://github.com/pommi/CGP.git* making sure we have root access to avoid mistakes.
6. We finally create the network of monitoring following the basic setup found in the following address: https://collectd.org/wiki/index.php/Networking_introduction

Once we have setup everything for the server to read and paint the diverse graphs we can finalize the Collectd configuration by creating the following configuration file in the image we have been developing so far:

```
root@:/etc/collectd/collectd.conf.d# cat network.conf
<Plugin "network">
Server "192.168.122.1"
</Plugin>
```

And update the file /etc/collectd/collectd.conf uncommenting the following line:

```
LoadPlugin network
```

And restart Collectd:

```
#: /etc/init.d/collectd restart
```

Make sure that the address of the collect hub is also included in the information found in */etc/hosts* and pay mind that the address seen in our configuration files is specific to our architecture, it will more than likely change if this configuration is done somewhere else.

4.2.5 Copy or Clone the file image for new VMS and configuration for cluster.

1. Once turned off, the image should be left out of virsh. To check if the image is safely deallocated of virsh, execute **virsh list –all** and the name of the VM should not appear. If it does, execute **virsh destroy *id_of_VM***.
2. Copy the image file into three different files (NameNode, ResourceManager and Workers) and keep the original intact for further use.

```
:$ cp hadoop_original.img hadoop_namenode.img
:$ cp hadoop_original.img hadoop_resource.img
:$ cp hadoop_original.img hadoop_worker.img
```

3. Change in the xml used for the original image the name of the file used for the hard drive for one of the new images (we start with hadoop_namenode) and let us create again with **virsh create *name_of_file.xml***
4. Once the image has booted, let us modify a few things before sending it to the front end of OpenNebula for instantiating them.

   - change /etc/hostname (only for the masters, slaves will be done with OpenNebula; ResourceManager should use hostname HopResourceMan as the terminology mark)

```
:$ sudo nano /etc/hostname
:$ Hop-NameNode
```

   - change /etc/hosts (only for masters again)

```
:$ sudo nano /etc/hosts
:$ 127.0.0.1 Hop-NameNode
```

   Once the images are finally completed with these last steps, poweroff the machines and make sure that they are out of virsh.
5. Now, OpenNebula can use both RAW and QCOW2 image types, QCOW2 though is a far less space consuming alternative so it could be a very good idea to change our images from RAW to QCOW2, to do so please do:

```
:$ qemu-img convert -f raw -o qcow2 –o compat=0.10
hadoop_namenode.img hadoop_namenode.qcow2
:$ qemu-img convert -f raw -o qcow2 –o compat=0.10
hadoop_resource.img hadoop_resource.qcow2
:$ qemu-img convert -f raw -o qcow2 –o compat=0.10
hadoop_worker.img hadoop_worker.qcow2
```

6. Once everything is configured, Hadoop in cluster mode can be tested manually using virsh before deploying it through OpenNebula; for a complete guide for doing this, please refer to ANNEX 1.
7. Finally let us send the prepared images to the front-end of OpenNebula so they can be deployed from there. Any method for sending them is accepted; we use scp to move files through ssh but if you have access to the physical machines you can move them in a USB or some other media format.

4.2.6 Deployment of images through OpenNebula

Once the images have been configured properly and sent to the front end of OpenNebula, we can start setting up the environment to instantiate and deploy the Hadoop cluster effectively via the front-end.

First of all, while not totally required, we should create a network that is specific to Hadoop so it will have a specific range of addresses where the cluster will look up. To do so, let us create first a

template file where the configuration for this network will be and then create such a network with the use of OpenNebula CLI (make sure you are using the oneadmin account).

The template file in this case is called priv.net and should contain something similar to the following:

```
#configuration attributes (dummy driver)
NAME = "Hadoop network"
DESCRIPTION = "A private network for Hadoop Cluster in OpenNebula"

BRIDGE = "br0:1"

#Context attributes
NETWORK_ADDRESS = "172.18.10.0"
NETWORK_MASK = "255.255.255.0"
DNS = "172.18.10.255"
GATEWAY = "172.18.10.255"

#Address Ranges, only these addresses will be assigned to VMs
AR=[
        TYPE = "IP4",
        IP = "172.18.10.100",
        SIZE = "50"
]
```

Take into account that the bridge option obeys the specific topology of the host we are using. Anyone should check their host (the one with the hypervisor for running the VMs) for their corresponding bridge interface, i.e. the one that the VMs created generally use. Again both the context attributes and the Address ranges correspond to the ips acquired from the bridge interface this of course will be different for everyone, the size can be as bigger or as small as desired, 50 is believed to be a good range for a big enough cluster setup.

To create this network we need to execute **onevnet create priv.net** and an ID corresponding to the network created should be the output. We take note of this ID as it will be used later on for the templates of the virtual machines.

Once the network has been set up, we need to create templates for the images we created so we can instantiate whatever number needed of machines. We then create three files that will have the following structure:

```
NAME = Hadoop_NameNode
PATH = /var/tmp/images/hadoop_namenode.qcow2
TYPE = OS
DESCRIPTION = "A hadoop used as NameNode for Hadoop on OpenNebula"
DRIVER= qcow2
```

Note that the PATH and NAME inside these templates should change for each file to reflect each of the images, also note that the path should be out of the home of the oneadmin user (/var/lib/one/ by default), such is the reason that the images are under /var/tmp/images. We will save these images with the following names: hadoop_name.one, hadoop_res.one and hadoop_slv.one.

Once they are saved, we can create them by executing **oneimage create *name_of_file.*one –d *ID_of_datastore_where_images_saved*** and as it happened with the case of the network we will get an output with the IDs of the images being created; should we forget the image ID we can execute **oneimage list** and all of the different images created in the different datastores will be displayed, also the state they are in. Before continuing you should check that the image state is **rdy** meaning it is ready for using as a disk for our environment.

Finally we can create the templates for the vms so they can be instantiated and sent to the host to be booted and managed by OpenNebula. As we did with the images, we create three archives,

though they will vary slightly, in general there will be two different seetings: one for the masters and one for the slaves. Therefore, the templates should look like this:

Masters

```
NAME = Hadoop_NameNode
CPU = 2
MEMORY = 2048

DISK = [ IMAGE_ID = 21 ]

DISK = [ type = swap,
         size = 1024 ]

NIC = [NETWORK_ID = 2,
        IP="172.18.10.102"]

GRAPHICS = [
        TYPE = "vnc",
        LISTEN = "0.0.0.0"]

OS = [ARCH = "x86_64"]

FEATURES = [ACPI = "yes"]

CONTEXT = [
        TOKEN = "YES",
        NETWORK = "YES"]
```

Change the NAME for the resource manager and also change the IP for a different one but inside the same network. The IP is used so these images always take these addresses while the workers will get an automatic one from OpenNebula using the network ID we provide.

Slaves

```
NAME = Hadoop_Worker
CPU = 1
MEMORY = 1024

DISK = [ IMAGE_ID = 23 ]

DISK = [ type = swap,
         size = 1024 ]

NIC = [NETWORK_ID = 2]

GRAPHICS = [
        TYPE = "vnc",
        LISTEN = "0.0.0.0"]

OS = [ARCH = "x86_64"]

FEATURES = [ACPI = "yes"]

CONTEXT = [NETWORK = "YES",
           TOKEN = "YES"]
```

The CONTEXT option will be the responsible of us being able to modify some options while deploying these images, NETWORK corresponds to all the options network related while TOKEN is used specific for OneGate (so VMs can push metrics to the OpenNebula frontend).

Save them under different names: vm_hadoop_nam, vm_hadoop_res, vm_hadoop_slv. Execute **onetemplate create *name_of_file*** and as it happened before the IDs for these templates should be the output. We will use this ID to instantiate the VM by executing **onetemplate instantiate**

ID_of_template, you can check the progress of the booting and copying of the VM by executing **onevm list** and in the end we should see a status of **runn**. Go ahead and instantiate one namenode, one resourcemanager and 3 workers for testing the Hadoop cluster by using the instantiating command.

4.2.7 Automatic configuration of Hadoop and testing the cluster.

In Annex 2 we explain what are the required configurations needed to start the cluster for the first time in a manual way, let us summarize the basic operations needed to be performed:

1. Change the hostnames of the workers accordingly. In the configuration we have done, the workers will have all the same hostname (Hadoop) we need to change this to different, in our case we used Hop-Worker-#. The network service should be restarted to reflect this change.
2. Both slaves and masters should have all of the information of networking in their corresponding */etc/hosts* file. This includes the hostnames and IP addresses.
3. Change the 'slaves' file of both masters, found in the configuration directory of Hadoop, and write in it either the hostnames or the IP address of all the slaves.
4. SSH should be passwordless
5. Once the NameNode is running, format it by executing *hdfsnamenode –format*
6. Once everything is configured, execute *start-dfs.sh* in the NameNode, this will launch the request to start all the different daemons for the HDFS component both in the master as well as the slaves. Correspondingly, in the ResourceManager we should execute *start-yarn.sh* and this will do the same but for the YARN component.

To be able to do this first automation of the cluster we supported ourselves in a group of bash scripts that will be executed when the machines are started for the first time that will make all the steps we have described before, showing all the different scripts could be cumbersome so an example of said scripts can be reviewed in Annex 3. In addition to the bash scripts used for automating the cluster we used two tools provided by OpenNebula, the first is OneFlow that let us create a multi-tier application called a service, in it we can define a JSON file that will have all the required information to instantiate all of the required components, in our case we want to instantiate one NameNode, one ResourceManager and 3 workers, to do this we produced the JSON file found below:

```
{
 "name": "Panacea Hadoop Cluster",
 "deployment": "straight",
 "description": "",
 "roles": [
  {
   "name": "Worker",
   "cardinality": WORKER_CARDINALITY,
   "vm_template": WORKER_TEMPLATE_ID,
   "elasticity_policies": [

   ],
   "scheduled_policies": [

   ]
  },
```

```
    {
     "name": "NameNode",
     "cardinality": 1,
     "vm_template": NAMENODE_TEMPLATE_ID,
     "parents": [
      "Worker"
     ],
     "min_vms": 1,
     "max_vms": 1,
     "elasticity_policies": [

     ],
     "scheduled_policies": [

     ]
    },
    {
     "name": "ResourceManager",
     "cardinality": 1,
     "vm_template": RESOURCEMANAGER_TEMPLATE_ID,
     "parents": [
      "Worker"
     ],
     "min_vms": 1,
     "max_vms": 1,
     "elasticity_policies": [

     ],
     "scheduled_policies": [

     ]
    }
  ],
  "ready_status_gate": false
}
```

As we see, it is very easy to create and add parameters to this JSON to create diverse set of 'services', we could for example add a new VM that could replace our current Collectd hub and we will only need to have a template and an image already defined in OpenNebula and it will be as easy as to add a new name with a cardinality and the template ID set in it. There are other configurations that can be performed using this tool that we are not using and so more information about OneFlow is available in the OpenNebula documentation that can be found in: http://docs.opennebula.org/4.12/advanced_administration/application_flow_and_auto-scaling/appflow_use_cli.html

The other tool we used was OneGate, this tool gives the ability to the virtual machines to send metrics and information to the frontend to do specific actions like scaling, this will be our cornerstone for auto-scaling Hadoop; more into OneGate can be found in the documentation: http://docs.opennebula.org/4.12/advanced_administration/application_insight/onegate_usage.html

In the end and to verify that the services are correctly running execute **jps** in the terminal of the different machines (we can access them by SSH), the output of this command will vary depending on which machine you execute it in. In general you will be seeing the following (numbers will change as they are random):

Hop-NameNode

```
3522 NameNode
6227 Jps
```

Hop-ResourceMan

```
3265 SecondaryNameNode
21857 Jps
21157 ResourceManager
```

Hop-Worker-#

```
9056 Jps
8935 NodeManager
2496 DataNode
```

If the configuration goes as planned and the services are running, we should be able to access http://hop-namenode:50070 (or the IP of the machine) and see 3 nodes under Live Nodes; this is the HDFS. We should also see 3 nodes at http://hop-resourceman:8088/cluster/nodes (or the IP of the machine); this is the YARN component.

To test that everything indeed is working run one of the examples provided by the version of Hadoop, preferably in one of the masters, by executing the following:

```
yarn jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.0.jar pi 32 10000
```

A job should appear under the applications section in the YARN component and when it finishes you should have a similar output to the following in the machine where the example was executed: **Estimated value of Pi is 3.14147500000000000000**.

4.3 TESTS PERFORMED

All of the tests performed in this section obey principles to test a diverse range of values to validate, as we have said before, the positive effect of the innovations given by PANACEA.

4.3.1 Related Work

Part of the tasks needed to accomplish the objective of this thesis besides the testing and final validation of the PANACEA innovations for the Cloud infrastructure correspond to the adaptation of a series of algorithms developed originally for Hadoop v. 1.0 and perform the needed changes to be compatible with Hadoop v. 2.x, concretely 2.6 which was the stable version of Hadoop and the one selected to be used for the testing environment.

The algorithms correspond to an internal proof of concept where implemented services provide different analytic algorithms for evaluating the statistics of values collected from sensors installed in the devices of a factory plant, (i.e. temperature, vibrations, rotational speed, etc.).

The execution of each of these analyses does not require a considerable amount of computational resources. However, when the amount of devices and sensors as well as the number of concurrent requests are growing, the resources should be increased in order to deliver the results in an acceptable time.

The solution we talk above consists of the different service packages as shown in Figure 7. The Device Monitoring service is a service providing a graphical user interface and continuously running some scripts to collect sensor data (consider that the sensor data will be simulated from real data to simulate different controlled conditions during the project). Then, there is a set of data analysis

services implementing the data analytic algorithms. Finally, all the analytic services share the Threshold Finder library which implements some functions used by the analytic algorithms.

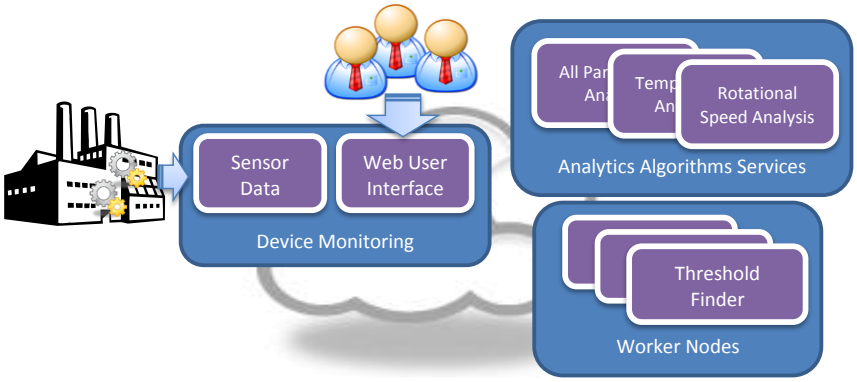

Figure 7: Block diagram of the solution ported to PANACEA

However, the final algorithms provided were not complete nor they were provided with their full-fledged source data, this due to being done by another division of ATOS Spain and shared with little jealousy. It is true that other, more advanced algorithms were being developed at the moment of this work and therefore could not be shared in the current circumstances.

Thankfully the adaptation can be done with little to no effort from Hadoop 1 to Hadoop 2 as its documentation shows, [53]. Hadoop provides binary compatibility between both, provided they do not use specific classes or methods that are deprecated, however in some cases even deprecated instructions can be used with just a warning that it is deprecated and should use a different one instead but continue running independently of it. However, and besides the changes been made to the algorithms two problems appeared, the need for the specific source data (in the format that the algorithm expected it) and the use of specific parts of the code that were either in packages not available or were codes with no apparent purpose as they were empty. As a result we ultimately took the decision that the use of the algorithms for this work will not be possible and another fast but usable alternative that will still give us insights about how to automate, scale and ultimately recover a cluster of Hadoop will be the use of the MapReduce example codes given by the framework for the testing of the use case as such it was decided and executed in this way. In annex 4 there is an example code of the algorithm change, where we comment the older code and we show where do we needed to make the changes to make this code compatible. In most of the other algorithms available for adaptation the changes were either the same ones or similar in context; we do not put the length of all the algorithms changed since it will consume too much space and will not be relevant to our case.

Among the other tasks for the goal set by this thesis, we include the development of images to be used as the backbone of a Hadoop cluster and make sure they are able to be deployed using OpenNebula and ultimately a study about which parameters could be used to evaluate the Hadoop cluster health and saturation as well as developing some metrics probes based on the finding from the study. The development of the images is thus documented in the following chapter where we specify all the steps for building the testing architecture which includes the development of the images, monitoring of metrics as well as the final usage in the OpenNebula environment. As for the study, this is intrinsically related to the results of the first phase of testing, internally known as the understanding phase, and as such it can be reviewed in chapter number six.

4.3.2 Tests Description

In a simplistic way let us describe the basic scenario that we will explore using our aforementioned environment: “A set of incoming series of tasks will increase in time the load of the Hadoop cluster. Hadoop will grow its number of node automatically, and de-grow them when the number of task reduces. We will measure when this happens and measure how much will take to perform those actions, if a system administrator had to do it manually. Also, we will measure the time to finish the tasks when a cluster is allowed to grow automatically vs one with a fixed number of nodes”. In particular we want to study this type of failures: DataNode problem, NodeManager problem, VM panics and restarts, DataNode or NodeManager fails, network becomes slow, one disk becomes slow, CPU/memory process hog and corrupt datablocks.

Of course, in the somewhat short timeframe of this thesis, not all of these kinds of failures can/will be studied. However, several of them are being simulated in our tests and later the information gathered from them used to train the Machine Learning Framework of PANACEA to detect them. After that, the necessary mechanism should be put in place to detect those failures and rejuvenate the failing node in a quicker way than the standard Hadoop installation.

Such tests were performed in two big phases, that we call the understanding phase and the knowledge phase.

- **Understanding Phase:** This phase corresponds to the first tests we performed, almost everything in it was manual, we just used a few shell scripts to perform routine work that in other cases will take too long to write or will be too repetitive. We used a first static cluster of 5 machines (2 masters and 3 slaves) that we make grow and de-grow in a semi-automatic way (using bash scripts). This phase will not only make way for better understanding of the cluster but to let us know what to really pay attention to, what are the steps to overcome the problems presented in these tests and also prepare the ground for better testing in the next phase were everything will be more autonomous and should behave as a human administrator is in charge of the challenges given by the cluster at any given time.
- **Knowledge Phase:** This next phase, we used OpenNebula as well as all the information we gathered from the understanding phase to actually verify that the innovations intended by PANACEA will help the DAaaS Use Case. In this phase everything is almost completely autonomous, with some basic intervention of the user for specific parameters such as starting the cluster, sending the jobs for Hadoop as well as providing evidence that the cluster is working as it was explored in the understanding phase.

Let us introduce now the rationale of both phases. In the understanding phase the most important thing is precisely understand the different steps we need to follow to provide Hadoop with elasticity, from an idle cluster all the way to the scaling of the cluster and the problems that we may encounter. Taking this rationale, we separate the tests of this phase in 3 big groups looking for understanding 3 big questions about the performance of Hadoop so we can then study the results in regards to the knowledge phase; the 3 big questions then are:

1. How does Hadoop behave normally?
2. How and when do we scale Hadoop?
3. What problems can Hadoop face normally and once scaling is performed?

The first question can be answered by looking at the most usual metrics, i.e. CPU load, memory usage, networking, disk I/O operations, etc., as well as the information provided by the GUI of the masters, i.e. the health of Hadoop. We need to see first how an idle cluster looks like and from there

start sending jobs to it to see how it reacts depending on the load provided; as we know how the load is been managed we might have a clue of when is a good time to start performing the scaling which leads us to the second question. The second question is actually two in one, to scale Hadoop we need to understand first how to scale both up and down on a “seamless” way, when performing a scale up we do not want to disrupt the work and when we perform a scale down we do not want to lose any information so we need to know what is the best way to do a clean scale in both directions; once we know how to do a clean scaling we need to decide when is the best moment to scale, based on what we found out in the first question. In a general way, the scaling up should be done when the resources are not enough to fulfill the work sent to the cluster in a given period of time, the QoS should be good enough so that all of the users have an answer to their petitions in a convenient time in which both users and provider agree upon. Using this we can also say that scaling down should only be done in the case that the cluster has been underused for a giving period of time, like after 10-15 minutes of being idle, it would be a good idea to shrink the cluster to its original size or to a smaller size that could still fulfill the QoS set by the providers for the users.

During and once we have answered the two questions with the set of tests performed, different problems are more than probably to arise, since the monitoring in this phase is done manually, we have to pay attention to the problems that we detect, either by looking at our monitoring as well as looking into the GUI of the masters as we have said before. Of course, we also need to know how to repair or try to avoid these problems; in fact, once we detect the most recurring problems we can then tell the proactive monitoring component of PANACEA what to pay attention to and how to repair it autonomously or even prevent it in a proactive way, it will depend of course in the severity of the error and the steps needed to be repaired or avoided.

Now, for the knowledge phase the tests that will be performed will be the most interesting ones from the understanding phase for scaling; as we want to test if what we understood and gathered from the first phase is applicable and can indeed provide enough expertise for auto-scale. However, we did a few speed tests to be able to compare the time it takes to start the cluster from the beginning as well as how fast it con provision all of the machines, all of the other tests performed have been selected specific from an array of tests from the understanding phase.

In all of the tests, as we said before, we used one of the example codes for MapReduce included as part of the installation of the Hadoop framework; with different parameters at given points to increase or decrease the load that the cluster will have to deal with so we can provide a more realistic scenario of how the DAaaS use case will be used by the prospect users.

All of the tests were performed in a space of time of nearly 6 weeks with some reworking of our architecture and tests in the middle, as we needed to provide that everything is running as expected and at the same time validating that it will actually be helpful for the final validation of the use case; so without further ado let us make way for the results of all the testing in the following chapter.

CHAPTER 5 – RESULTS AND LESSONS LEARNED

This chapter as it name says, will present the results obtained during the 6 and so weeks that we performed the tests devised for the understanding phase as well as the knowledge phase. We will also introduce a section when we give a summary of the lessons we have learned during the tests, including punctual problems or thing to take in consideration for anyone wanting to replicate the results or even try the Hadoop panorama.

5.1 Results for the Understanding Phase

As we said before, the tests performed during this phase will help us understand the operation of a Hadoop cluster, both from the static point of view as well as the elastic point of view so we can put all of the knowledge gathered through this phase in the components of PANACEA required to make use of the proactive and autonomic innovations for the particular DAaaS use case. Let us remind the reader that in this phase we use a semi-static Hadoop cluster comprised of two masters and 3 slaves (Hadoop Workers) and our monitoring server is the physical host in charge of hosting the virtual machines that conform said cluster.

Before we even set foot on describing the different tests performed, let us comment a brief configuration we had to make. In the default setting, Hadoop will utilize a FIFO scheduler which is quite good as it will try its best to finish each application it sends as fast as possible. However, this is in many cases very inefficient as it may be unable to process too many apps at the same time and thus act as a bottleneck for processing applications in the cluster. Thankfully, Hadoop provides another kind of scheduler, a fair one, which tries instead of letting as much applications as possible to enter the cluster to be processed at the same time; sharing the resource 'fairly' between them. While this might hurt the final processing time of the different application that are being processed in the cluster, it will lower the time that the applications will stay in the system waiting to be processed. This scheduler is already configured in the description we gave in the chapter dedicated to it and its clearly commented in the corresponding example, so no further explanation for configuration should be needed.

5.1.1 How does Hadoop behave normally?

The first thing to revise once the cluster is working is how it behaves on an idle state, we know for a fact from the Hadoop Wiki, [56], that some checks are being done by HDFS as well as YARN in the form of 'heartbeats'; this heartbeats could probably affect both the networking as well as the CPU of the masters as well as the workers.

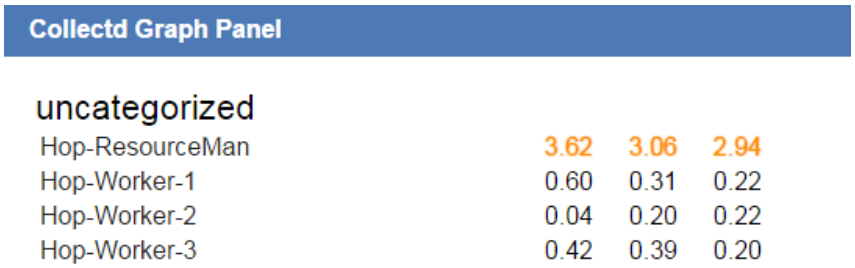

Figure 8: Idle cluster of YARN

The figure above shows the CPU load of the YARN master as well as the 3 workers. The first column represents the load in the actual moment, while the second column represents the load 2 minutes

ago while the last one represents the one found 5 minutes ago; in general any number above 1 in the CPU load of a Linux machine represents that the CPU is being high loaded. In the figure we can clearly see that while the workers do not reach the number 1, they are clearly some operations being done at a given time; the load in Worker 2 was a little high 5 minutes ago (from the time we took this sample) and it is currently without that much load, however that is not the case of workers 1 and 3 since both had lower loads and now they have increased. After a few more minutes we could see that the load becomes at some point 0 for all of them but they increase slightly after 2 minutes or so.

We have seen how the specific operation of Hadoop may affect slightly the reading of CPU load, but now let us see how the CPU is being loaded when we sent jobs to be processed by the cluster. We performed 3 different sets of testing which we run up to 4 times to be completely sure that our results were as statistically correct as possible; the sets are composed of 10 applications at once, 5 applications at once and 2 applications at once all of the applications make use of the pi MapReduce example code with 32 maps and 10,000 samples per map.

Figure 9: CPU load for 10 applications at a time

As we see in Figure 9, the load goes up quite rapidly when we sent 10 apps at the same time to be processed by our cluster. Something that seems obvious, could be very deceiving as we could expect that the more apps we sent to the cluster the more loaded it will get but that could not be the case in some exceptions. We executed 4 sets of this test and we found that the load is quite similar between executions, as the time of executing is also similar (ranging from 20 minutes to 37 minutes). Let us execute 5 applications at a time

Figure 10: CPU load for 5 applications at a time

In the figure above we see that CPU load when we send 5 applications at a time is a little less stressing to the one we found in Figure 9, however in some cases it was very similar to the former and as the figure shows in some cases it was more loaded as in the case of the Hop-Worker-3. Let us make a visual comparison between both set of tests before continuing.

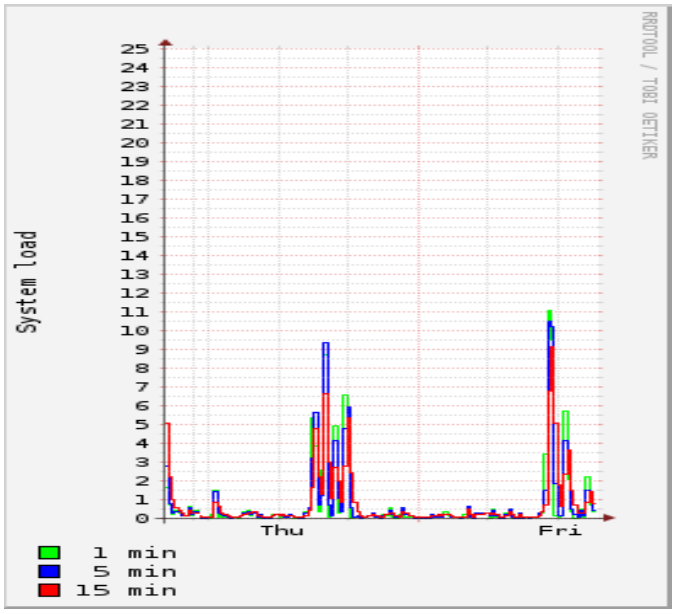

Figure 11: 10 apps load vs 5 apps load through time (Hop-Worker-1)

In the above figure we can see a comparison between the system loads of the 10 apps at the same time (middle) with the 5 apps at the same time (right) and we can clearly see that the loads between both of them are very similar. Making this comparison, we did not expect that the final test with 2 apps at the same time will be any different, and it was not as the figure below shows.

Collectd Graph Panel

uncategorized

Hop-ResourceMan	5.91	3.63	2.60
Hop-Worker-1	9.29	3.69	1.41
Hop-Worker-2	8.77	2.80	1.06
Hop-Worker-3	12.90	5.16	1.93

Figure 12: CPU load with 2 apps at the same time

With this amount of evidence we believe we can say that the amount of apps at the same time cannot be used as a metric for scaling as it does not vary much with the increase of number of applications in the cluster at any given time; so we can only use it to asses that the cluster is really working. However we will still be paying attention to it since it may not be a good metric for scaling but it could be for a detection of a problem, i.e. one of the worker machines is highly loaded while the others appear to be idle.

Memory and networking are other metrics that we could consider, however seeing how the CPU load behaves, we would expect similar results, as for the networking it is a similar case seeing as Hadoop is a distributed framework and therefore it requires a lot of network communication while idle (as we have seen at the beginning with the heartbeats) as well as in a running state. In the following image we see in just one graphic the memory usage of the tests performed before.

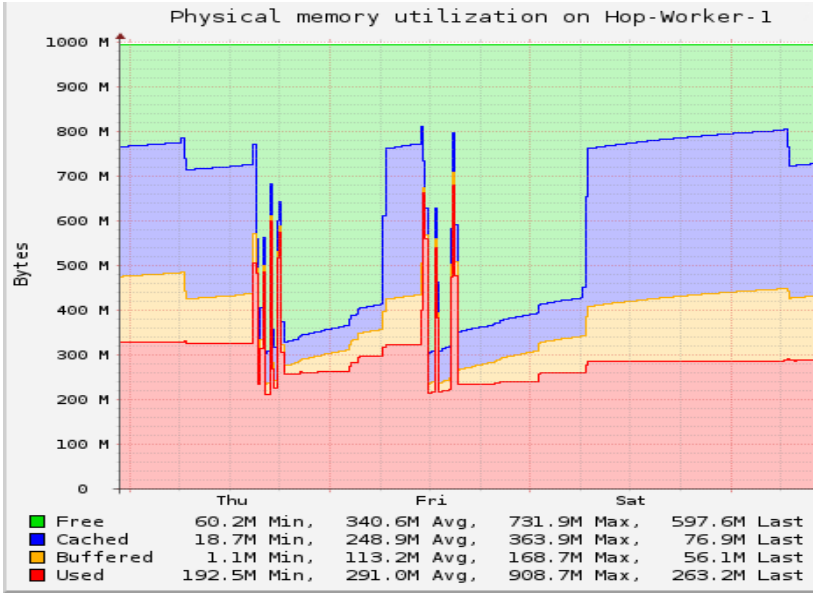

Figure 13: Physical memory usage during tests

As we clearly see, there exists a direct correlation between the CPU load and the memory usage. In general terms it is quite the same with little use while idle and a heavy use while performing the required operations of the MapReduce applications. As with the case of the CPU load, this are not very indicative of a need for more resources, it merely shows that the cluster is being used. Now, regarding the networking and seeing how Hadoop uses both CPU and memory to a very high use, we could look forward to seeing a similar behavior with the interface in charge of the network. A figure, showing the traffic of the network interface in some of the different tests performed during this phase.

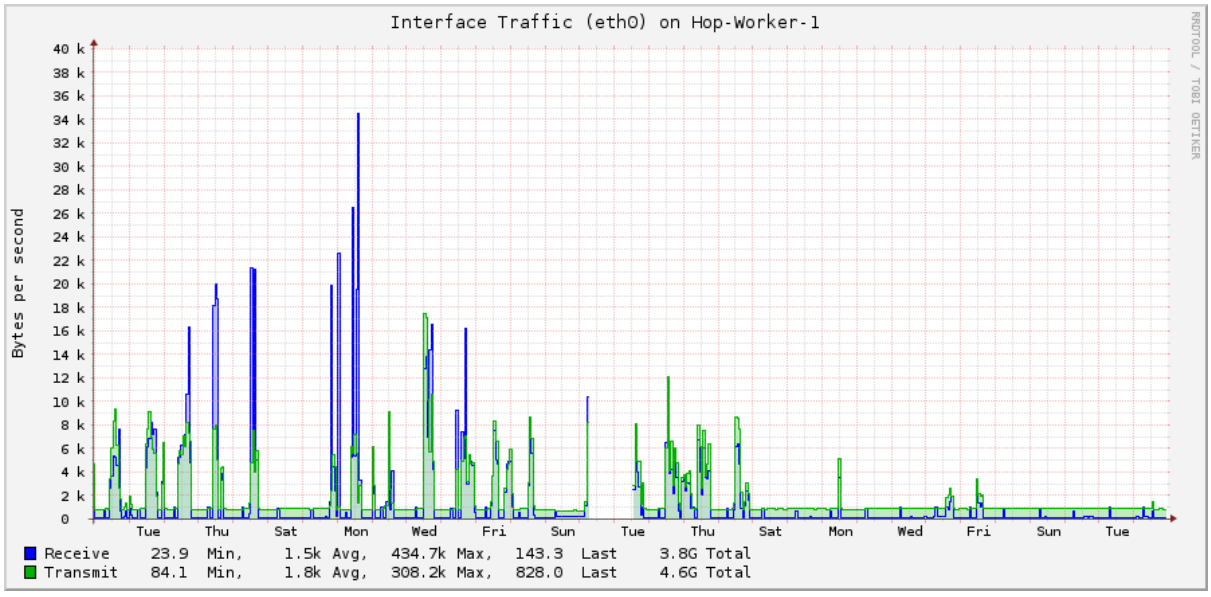

Figure 14: Interface traffic during a set of tests

Seeing now these three evidences we can conclude that Hadoop tries to use the maximum capacity available in the different worker VMs to deliver fast and reliable results to the users. None of the commented metrics may be used as an indicator of a need for scaling, except perhaps when the network usage is too high as it seems to appear only when a high set of applications are able to be executing in the cluster.

As we go away from physical metrics we consider that time is probably one of the most important things to consider when scaling and probably a much better fit to our case, we should expect that when we increase the amount of available resources, the amount of applications that we can receive will not only increase but we should also be able to finish and deliver the applications in less time. Before scaling we need to know for a fact how much time will it take for a normal cluster, i.e. our static cluster, to complete a set of given tasks and with this information we can then compare it against our results when we begin the scaling processes. For the first test in this direction we repeated the test of 10 apps at a time, we hope to look into deep into it to understand just how much time does the cluster need to be able to finish executing this 10 apps. We performed this test up to 5 times, and only one of the results is examined in the following figure.

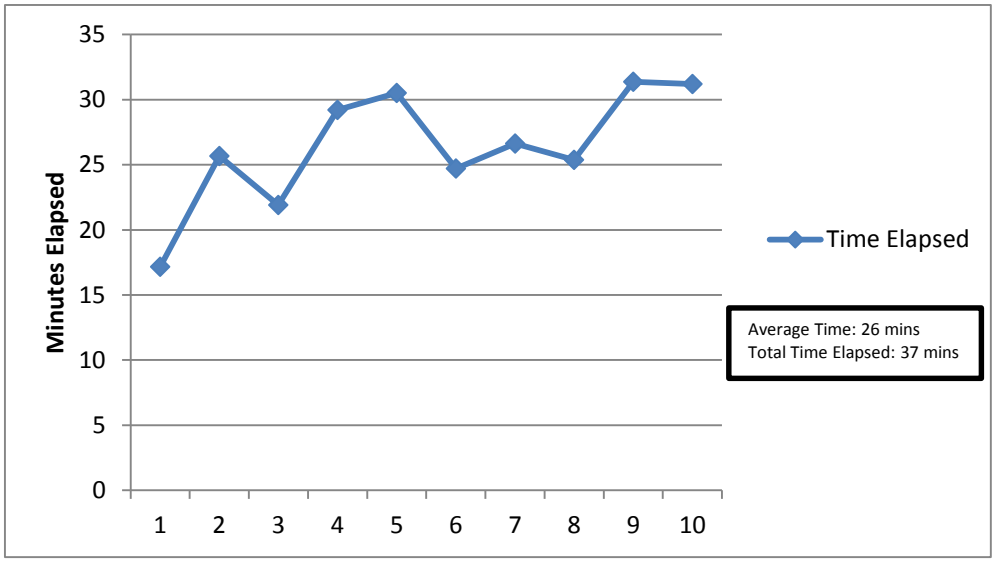

Figure 15: Elapsed, average and total time for 10 applications in cluster

In this figure we can appreciate the different times that each application took to be completed, the average time is in red and is about 26 minutes with the total time elapsed for this specific test in green and with 37 minutes. While this result is a good one and the other tests got similar times, between 33 and 37 minutes, we consider this to be a very optimistic setting since we only executed this 10 apps and waited until they finish; we appreciate the result but in a real case scenario the chances of having just 10 applications running or at least with this 'size' of applications are very slim so a more 'stressful' test needs to be executed. This new test correspond to the execution of 10 apps at a time, adding 10 more apps every 10 minutes during 40 minutes, for a total of 50 apps total. If we extrapolate the results from the first test we should expect a time of finishing of around 3 hours. The results are displayed in the figure below.

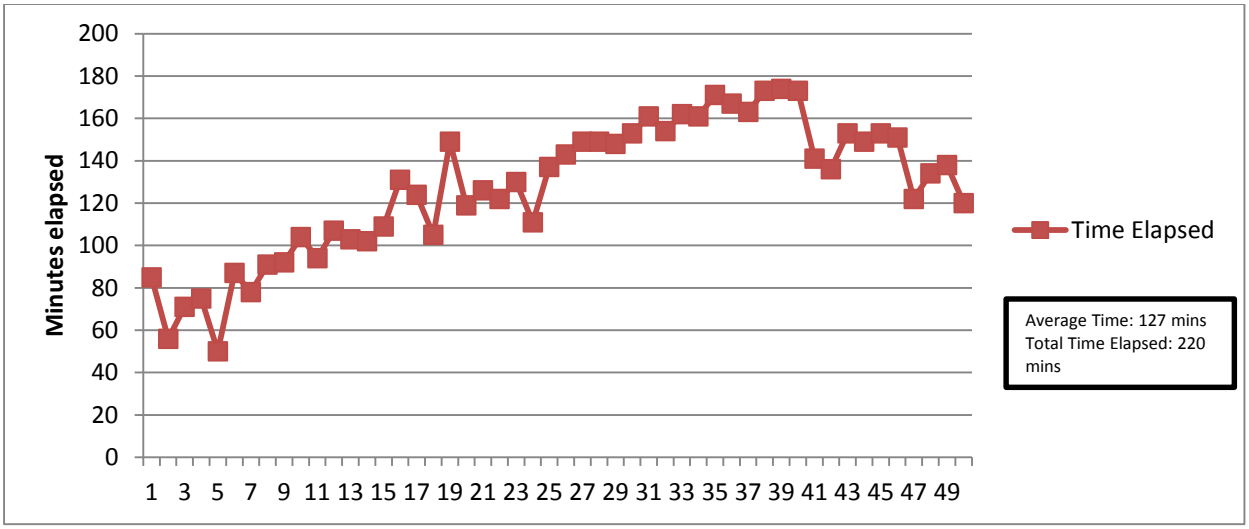

Figure 16: Elapsed, average and total time in minutes for 50 apps sent in 40 minutes

In the figure we can see that the average time in this case per application is of about 2 hours, which we consider is not a good time since in the other case we saw that the average time per app was of 26 minutes, however the total time is quite near what was expected with 3 hours and 40 minutes total. As with the other tests performed, this one was executed several times with similar results; but we found that in most cases a random number of applications could not be executed or were actually lost, more about this in the problems section. In addition to this test, we did perform two more tests with the same structure, we performed a 70 apps test in an hour (10 apps each 10 minutes) however this caused a lot more problems than the 50 apps test since it got reduced to 40 to 60 apps each time of execution, clearly this is the best example that the cluster needs to scale up in order to avoid this. In contrast we also performed a smaller test with 30 apps in the original time, 40 minutes, and the results can be seen in the figure found below, in it the final elapsed time was of 1 hour and 37 minutes (97 minutes) and the average time was of 49.5 minutes, much more decent than the 2 hours that we reached in the 50 apps test, it is however a big enough elapsed time to consider scaling up and it will definitive depend on the service level agreements (SLAs) that the platform and the user agree upon.

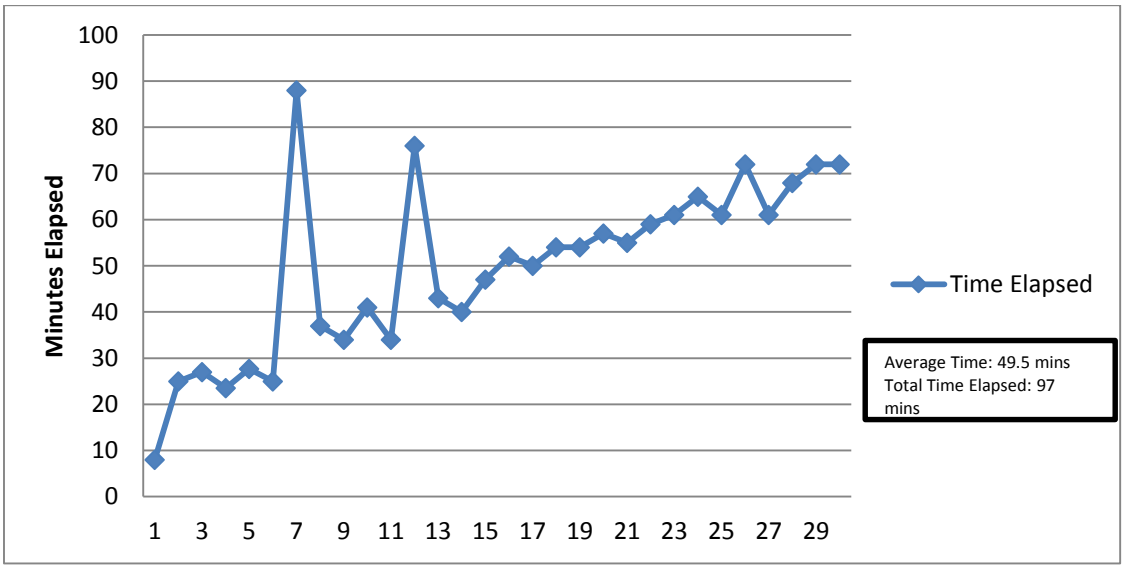

Figure 17: Elapsed, average and total time for 30 apps in 40 minutes

Since we have tested with the same exact parameters for all the applications in the last few tests, as part of the experimentation we decided to test how much time it will take for the cluster to answer 16 different applications and how does it affect in the completion time of each of them and from that point we could test increasing the amount of applications to see how this affects the time required by the cluster to finish the execution. As with the other tests we used the same pi example code but reducing or increasing the parameters to have less or more complex applications, like we will have in a more real setting. In the figure below we proceed to examine the results of said test.

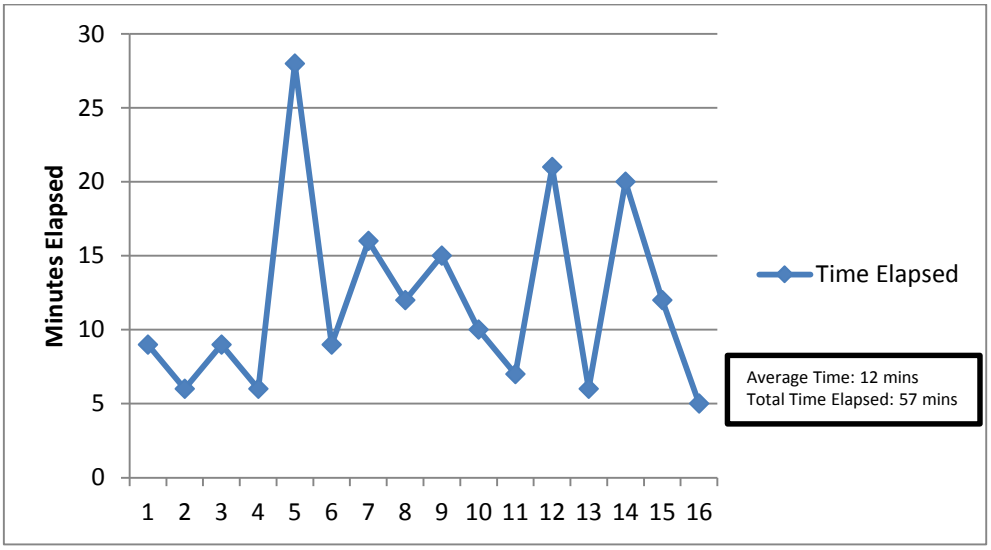

Figure 18: Elapsed, average and total time of 16 apps with different sizes

Interestingly enough the average time in this case is of near 12 minutes, with different times of execution, depending on the complexity of the problem (the most demanding app required 28 minutes while the less one took about 5 minutes) but also on how much resources were available, i.e. all of the less complex apps manage to finish before the more complex one releasing resources for the others in execution thus reducing time of execution. The total time did not escalate too much, 57 minutes, in comparison with the 10 apps test; however, other tests got as much as 1 hour 27 minutes with the less time of execution around 48 minutes.

We have talked a lot about the metrics we can easily have in any system for testing purposes. However, Hadoop have its own set of metrics available that we can use to check specific parts of both HDFS and YARN. For example, in HDFS we can have the block count, the percentage of minimal replicated blocks or the amount of disk space still available for HDFS among other metrics, while in YARN we have the number of containers, the status of the applications in the cluster or the amount of applications waiting to be processed, and as with HDFS it have other metrics. Both HDFS and YARN already, and by default, make some checkups on a regular basis: the 'heartbeats' we have talked about before. This 'heartbeats' are very important for both components of Hadoop since they basically tell the master that the worker is functional and available, it also carries some information like the number of blocks that they have or, if there are applications running, what is the application running in a machine and what resources are they allocating.

With this in mind, trying to perform a double check on this will be duplicated work and we should check other more important facts like the space left for HDFS, in our tests we never exceeded it but in a real case scenario we are talking about terabytes of information that need to be processed and in some cases even saved so controlling this and adding the appropriate resources will be appropriate. Other example could be the applications waiting in queue to be processed by YARN, if

we detect that the number is increasing or has not been reduced in a long period of time or say, by looking at the amount of applications that are in the system by knowing beforehand what is the 'sweet spot' of applications in which we can deliver good results with the amount of resources we currently have and if this is bigger we can proceed to scale to arrive to that 'sweet spot' again; this, as you might guess, requires a lot of preparation beforehand.

As we said before there are other interesting metrics given to us by Hadoop itself. While they are already being controlled in some form, one interesting metric that may help us understand when is a good time to increase the resources we have available in the cluster or with a combination of idle let us know when to reduce the amount of resources available, in this way achieving total elasticity for Hadoop is the metric that let us know how many MapReduce applications are currently in the queue, i.e. waiting to be processed. As the amount of queued applications increase, this means that more and more users/systems are waiting for a response; in general we will like to have this number as small as possible or within a given threshold so as to be able to provide an specific QoS. To test this we decided to look first at how the queue behaves when there are only 3 nodes in the cluster, once we know how it behaves we can then compare it to how it behaves when we add more resources in the scaling section. We performed two different tests, one with 30 applications, 10 each 10 minutes over a period of 20 minutes and then one with 5 applications each 10 minutes over an hour for a complete set of 35 applications. Let us see the results in the following figures.

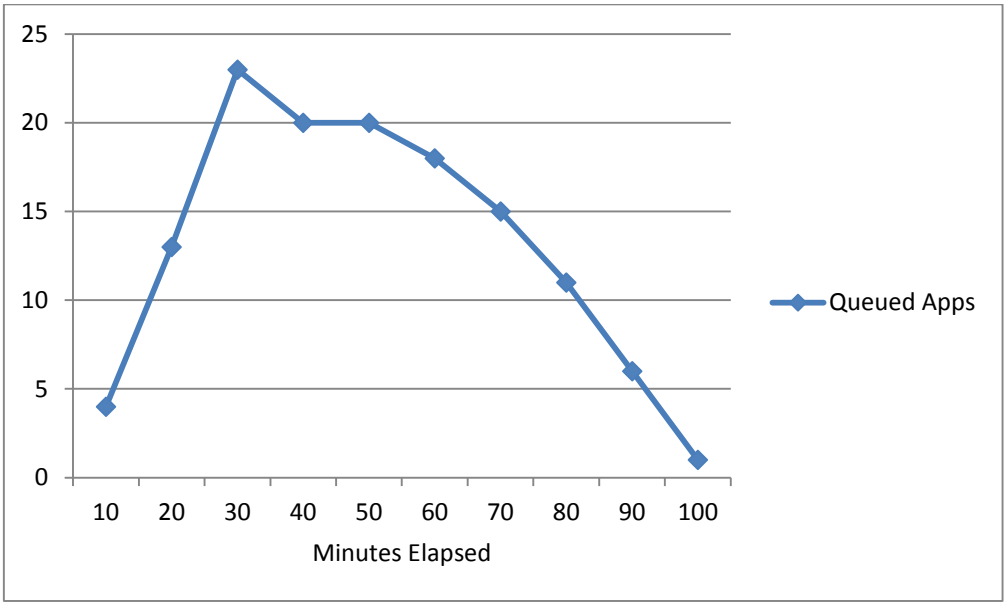

Figure 19: Queued apps through time for 30 applications over 20 minutes

In the figure, we can clearly see that since the first 10 minutes we already have at least 4 applications waiting in line and this only gets worst as time progresses, it is clear that the cluster seems to be insufficient in this case as it reaches almost 23 applications in queue once we have finished sending all 30 applications, that is that in that time it has only been able to process 7 applications at most. Once we have finished sending application though, the queue starts reducing in some steps. Based on this first finding we are clearly sure that as long as we continue sending more and more applications that cannot be processed the queue will grow indefinitely. So let us see in the figure below how did it manage with 5 apps each 10 minutes over an hour.

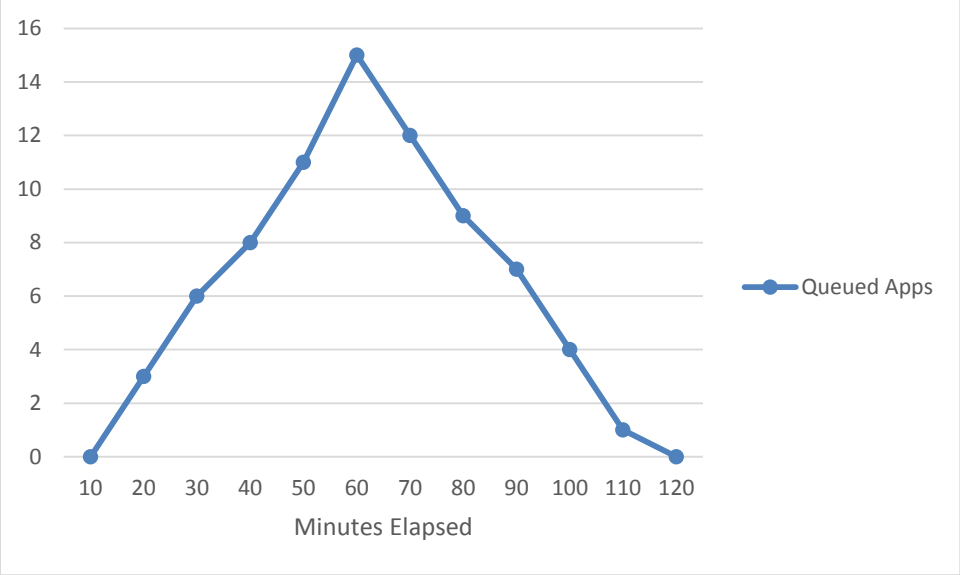

Figure 20: Queued apps through time for 35 applications over 60 minutes

As we though before, in the figure above we can clearly see that while the queue do not grow too much, it continues to increase quite rapidly. In the beginning we do not have any applications in the queue as it seems there are enough resources for processing 5 applications concurrently, this rapidly is lost as more and more applications come and are unable to be processed. However once it finishes and there is enough room to process all the remaining apps this number starts descending, so it seems clear that as long as there are enough resources we can decrease the amount of applications in the queue in a given time, if we add more applications to the system we need more and more resources so the queue do not grow too much which will represent that a group of users is waiting for processing, another issue will be how much time will they wait. We should expect a similar behavior for the scaling process but we should see it more rapidly.

After much discussion, it seems that the most appropriate metric available to be used for scaling is the number of queued applications at any given time. However, there are two main courses to take into consideration with this metric, one is time and the other is the amount of queued applications. We can set one or two different thresholds using any of this two courses, we could set, for example, a threshold for waiting time in which after a giving time if the queued applications have not been reduced or the time for the running MapReduce applications have exceeded a certain threshold of time we scale up or in the other hand if a threshold of number of queued applications waiting have been exceed then this will be our cue for scaling up. Using the same logic, we could use a threshold for scaling down in which if the cluster has been idle for a given time we can reduce it or for example if we have excessive resources to finish the remaining. Any of this different courses or a combination of them seems like the best alternative for scaling and in return make Hadoop elastic, we continue the tests using this newly found knowledge now for scaling and the results are exhibited in the next section.

5.1.2 How and when do we scale Hadoop?

Elasticity, as we have said before, is the ability of a system to automatically provision and deprovision computing resources on demand as workloads change, this means scaling up (add more resources when required) and scaling down (reduce resources when they are not needed anymore). However, there are two important things to consider when scaling, especially when scaling down, and that is: the configurations needed to be done inside the VMs so that cluster can accept the new

resources in a transparent manner and the persistence of data in the VMs. For the first part, most of the configurations in our case are done, including being able to SSH the different machines without a password and also having the same version of Hadoop including the same configurations. Now for the persistence of the data, there are some considerations like the kind of data we are managing as well as the kind of service we are operating. There are two routes:

A) We do not care about the persistence of data: in this case we call this a forced shutdown, in such cases we do not perform any recovery of the data and we just eliminate, destroy or shutdown the VM we are not using anymore without taking measures, this in determined scenarios cannot be done without repercussions.
B) We care about the persistence of data: this is our case and we call this a 'graceful' shutdown. We need to make sure that we do not lose any information as it may be critical for our users/customers, so we need to make the required steps to do so.

Now that we have understood the different things that occur when the cluster of Hadoop is being used and we have decided on the parameters (metrics) that we need to pay attention to, meaning that the resources are not enough and in need of scaling we can concentrate on the different tests to scale Hadoop, this making it elastic.

The workflow for scaling up Hadoop, as we have looked up, is quite simple and it is defined as follow:

- Make sure that the hostname of the new machine is different from other in the network/cluster.
- Add its hostname or IP address to the 'slaves' file of the masters; also make sure that it has a route setup in the */etc/hosts* file so it can reach it.
- Make sure that all the current slaves also have the route to reach this new VM.
- Run the slaves daemons in the new machine.
- Wait.

As we can see the scaling up action is quite simple, and in fact for the YARN daemon it takes less than a minute to be up and running and already providing resources to be exploited by the cluster while HDFS takes around a minute and a half to be up and running. Something to take into account for HDFS is that when the scale up is performed, the newly added VM does not have any HDFS blocks and will not have until an application is executed while this new VM is available, so in the case that the added resource is permanent, a new task called balancing should be performed, once this balancing task is performed the NameNode will issue a directive for the workers to coordinate that some replications of blocks be sent to the 'new' VM and also delete any over-replicated blocks that are in the cluster once all of the workers are 'balanced'.

Let us focus now on the workflow for scaling down 'gracefully', as with the scaling up Hadoop makes it simple enough but it requires time to be able to do so, and is defined as follow:

- For both HDFS and YARN, add the hostnames or IPs of the VMs intended to be discarded (scaled down) in the 'excludes' file of HDFS and YARN.
- Once the above is done, execute the '-refreshNodes' operation on the HDFS and YARN respectively.
- Wait for the process of decommission (scaling down) to finish, especially for HDFS. During this process YARN will wait for the containers of the decommissioned node to finish and then will terminate the daemon, HDFS for its part will replicate all the information in the decommissioned node, will not send more information to it and once the process is finished it will mark it as decommissioned (it will not stop the worker daemon).

- Once decommission is done, we can safely eliminate the VM from our Cloud infrastructure.
- We should eliminate all traces of the existence of the decommissioned VM once it has been eliminated from our Cloud. This includes eliminating the routes, the hostname/IP from both the 'slaves' and 'excludes' file and finally perform a 'refreshNodes' operation again.

As we said before there are several strategies that we could take to make the escalation process, but we have decided to set out efforts in the applications currently waiting in the queue to be processed, let us then proceed to show the tests and results acquired during the escalation tests.

Before even trying to compare our results from our static cluster to the one we are about to scale we performed a set of a small test just to confirm that some of our findings and conclusions about Hadoop were correct. In the following figure we show the CPU load of fully fledged 6 worker node cluster with only 2 applications of MapReduce being processed at a time, we performed this test 3 times with similar results to said figure.

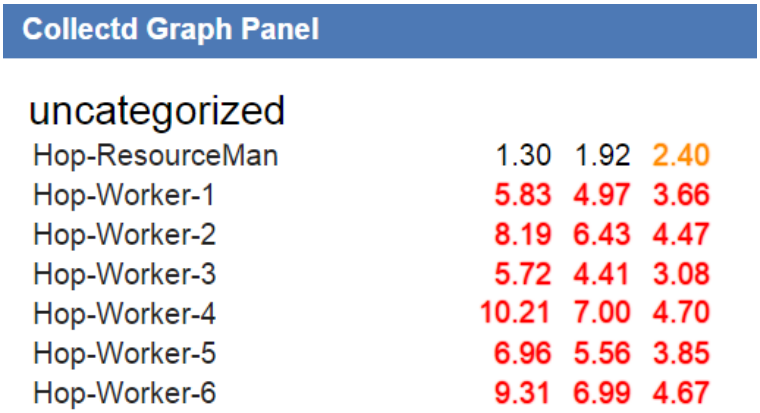

Figure 21: 2 apps at the same time in a cluster of 6 nodes

As we can clearly see from this small set of tests, our first impressions about Hadoop were correct, the cluster will try to push all the resources to their maximum capability to deliver a response in as less time as possible thus confirming that the CPU load will mainly be used to assess that the cluster is working and could be used for scaling down when a threshold of time in an idle state has been reached.

Before going into the details of our testing for this new section, we desired to confirm how the cluster will behave according to the selected metric of queued applications. We used the same two tests, we performed on the 3 node cluster and looking back at our results we decided that it would be best to add 1 node each 20 minutes, up to a final cluster of 6 nodes in the cluster. We trust that in this lapse of time we will be able to see how much the reduction is possible. Let us see below the figure corresponding to the first test of 30 apps in a 20 minute lapse.

In it we can see that while it shows a similar behavior as with the 3 node cluster, the number of queued applications after we have finished sending applications to the cluster is quite less than the other, 19 vs 23, which is 4 applications less; meaning more applications can be processed and we should see a better QoS as a result of it. Not only that but the speed in which the queued applications are being processed is significantly less, getting to zero queued applications 20 minutes before than in the 3 node cluster. Using this result as evidence, we can say that if we add more and more resources (nodes) we are sure to see less and less applications in the queue, or if we add more applications the queue will not grow so fast than in the 3 node case. Of course, there are other things to consider like how big we can grow the cluster or how many apps are we willing to have waiting in the queue.

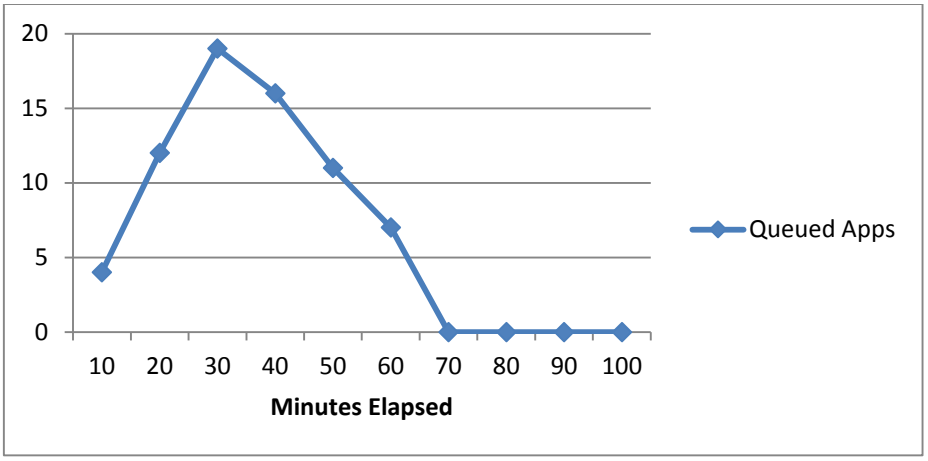

Figure 22: Queued applications for 30 applications in 20 minutes for a 6 node cluster.

To finish and have more evidence about how scaling up affects our queued applications metric, let us comment the figure below that shows the results for the test belonging to the 35 applications during one hours, sending five each and every 10 minutes.

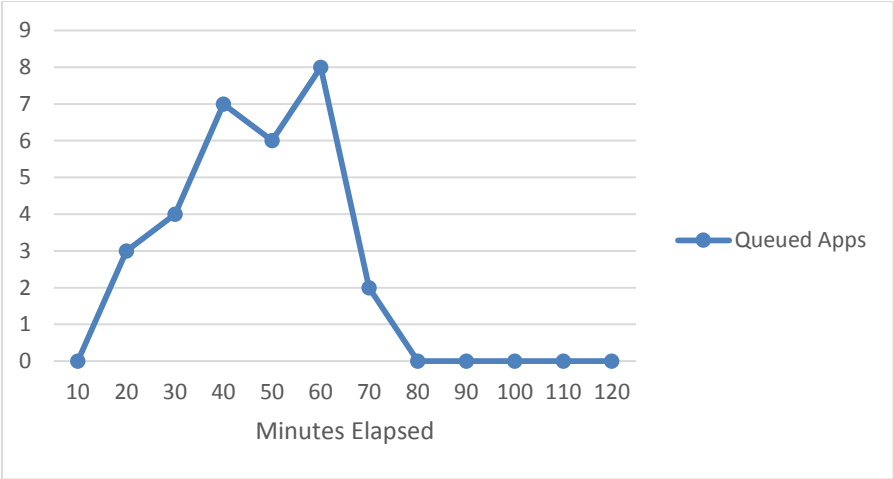

Figure 23: Queued applications for 35 applications in an hour for a 6 node cluster

Similarly to the one before it, we see an increase in the number of applications in the queue; however, the number of applications is increased quite slowly and in some points descending even while we are still sending applications to it, but it clearly corresponds with the time we are adding the different nodes to the cluster. It also confirms that with enough resources and even if we have a steady number of applications being sent to the cluster they can be processed and even the queued apps be reduced if not by a great number just enough; it will always go back to what QoS we desire to have. Once we have understood this we can now see how this results affect our tests about the time of execution and total time of the applications.

We did a lot of testing during this stage but in the end most of it was deemed either inconclusive or insufficient since it got a lot of errors or was not showing good enough results to be considered as conclusive. Such tests include the following:

- Sending 16 complex apps and adding just one new node after the queue of apps was stuck in more than 5 apps (about 20 minutes). This was insufficient as the total time of execution between the 3 node cluster and the 4 node cluster was not reduced drastically, having even some similar times at some points.

- Sending 50 apps in a time lapse of 40 minutes (10 apps each 10 minutes), adding 1 node after there has been 30 apps in the cluster. In the different iterations of this test we got a high number of failed applications of this time, thus we decided to reduce the app count.
- Sending 40 apps in the same lapse of 40 minutes but adding two nodes, one after the first 30 apps and another one after 5 minutes. Inconclusive, same results as the test of 50 apps.
- Sending 70 apps in a time lapse of 60 minutes (10 apps each 10 minutes) adding 1 new node around each 20 minutes considering the queued applications at that time. As with the case of the 50 apps test we got less apps in each and every iteration of the test so we finally we decided to perform a similar test with less applications.

The most prominent of our tests and the one we are showing the results comes from the reduction of the original 70 app tests in an hour which got reduced in half for a total of 35 apps in an hour, 5 apps each 10 minutes, and including a new node around each 20 minutes. We firstly tested how long will it take for a cluster of 3 apps to process this 35 apps so then we can compare it, as with the other tests we performed this same test 4 to 5 times and the figure below shows the results of one of this first test.

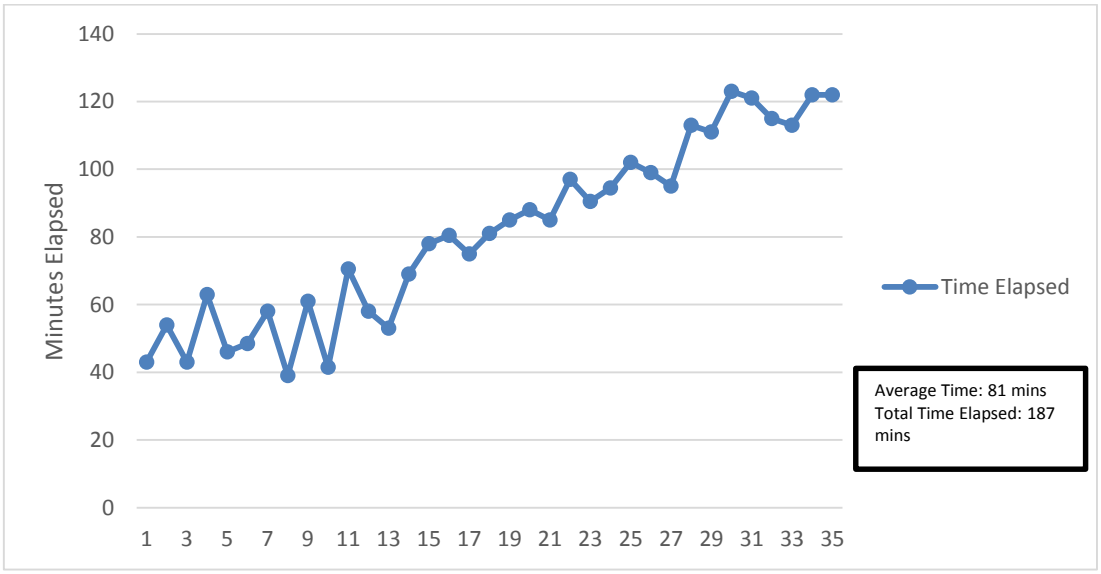

Figure 24: Elapsed, average and total time of 35 apps on a 3 node cluster

As we can see from the figure, the total elapsed time as well as the time elapsed per app increases rapidly in our 3 nodes without scaling. With an average time of 81 minutes we could not consider this a good time for delivering responses in a use case such as this; the total time elapsed is too high as well ramping to 187 minutes this time of course will most than surely cost a lot of money in a production environment. Now that we know how a cluster of 3 nodes will deal with this let us see the following figure with the results of the testing that got up to 6 nodes in the cluster.

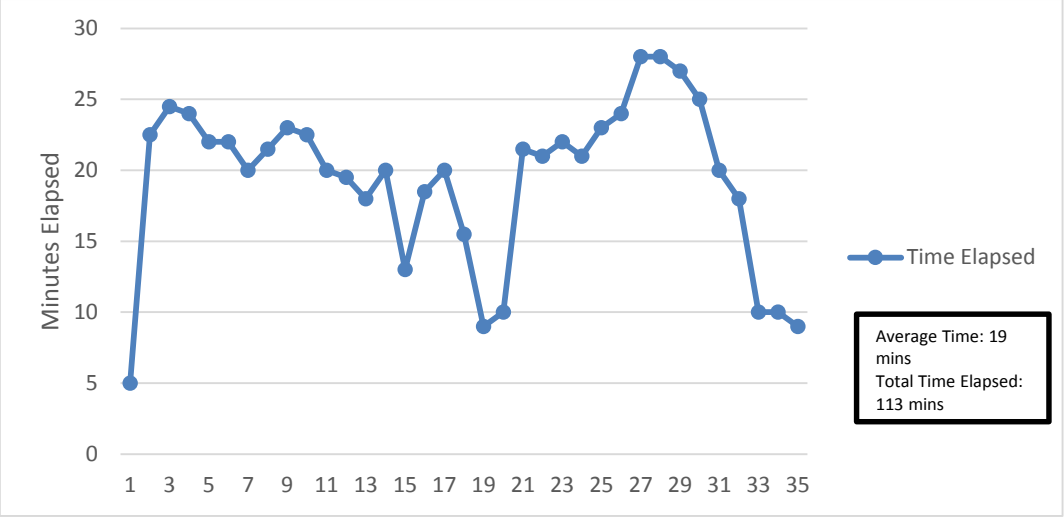

Figure 25: Elapsed, average and total time of 35 apps on a 6 node cluster

The expected behavior when adding resources to the cluster was present in the results of the 6 node cluster test, while the total time elapsed was only reduced in 74 minutes which is good as itself, the most prominent result is that of the average time of execution per application of barely 19 minutes. This is a very good result, but we would have liked to reduce the total time of execution as well but further testing need to be performed.

While we only did performed testing up to 6 nodes in the cluster mainly because doing most of the hard work manually is too cumbersome, this results gives us enough material to devise better and more automated tests for the next phase of experimentations. With some automation of the tests we are sure to process more complex test and thus finding better and more interesting results.

5.1.3 What problems can Hadoop face normally and when scaling is performed?

The purpose of any test is to detect as many problems as possible beforehand so they can be fixed if they can be fixed, avoided if they can be avoided or prepare the systems if they are inevitable. Hadoop is not a silver bullet and the use case as mentioned before do not aim to provide only Hadoop, but is definitively the software foundation needed for other solutions to be set in place, so we need to make sure that we can prevent, avoid and/or recover from any problem that Hadoop may face that could jeopardize the entire Cloud infrastructure and the services provided by it.

Although we concentrated our tests in the YARN component of Hadoop since it is the one in charge of the MapReduce management and execution, HDFS plays a very important role in any of those tests as it preserves not only intermediate results but also the final ones of each and every application sent to YARN. So in Hadoop we have two kinds of problems:

1. The ones related to YARN
2. The ones related to HDFS.

YARN is in charge of managing the MapReduce applications that arrive to the cluster; in normal circumstances the ResourceManager will designate an AppMaster from the pool of workers and this will be in charge to coordinate all of the workers and their resources (in form of containers) for delivering the final result to the ResourceManager that in turn will return it to the user. Having said that we detected two main problems: The application failed, it got accepted, attempted to perform its actions up to 2 times but with no response; on the other hand, any of the 2 different operations, map or reduce tasks, may fail and in this case the application will be processed but ultimately fail. The source for both errors seems to be a problem with networking, either because it could not

communicate with the other workers or if the workers seemed to be too busy to respond the different request through the network; as a result, both errors may appear independently if there are performing an scaling operation or not. This is in general the problems we managed to encounter in the YARN component.

HDFS is only in charge of the storage of information and as such we expected it to be less prone to failures than YARN; not so, as a matter of fact this was the component that gave us more headaches during testing and we list the main issues we encountered while using HDFS:

- At random, an application from MapReduce could not write because it could not be replicated in any of the nodes of the cluster. The main cause of this was traced to networking issues or too occupied node.
- HDFS caused a lot of issues when scaling down, generally it will perform the decommission process fairly well and in time but in random scales it will get stuck in the process and refused to move; even when scaling down when everything was idle.
- At some point the under-replicated blocks were setting alarms and hindering the performance of the cluster; this mainly caused while not performing the scaling down correctly (waiting for decommission process to finish, having less than the minimally replicated blocks).
- With more and more nodes being scaled the number of different options available for writing into were more but instead of being helpful it seemed to hinder the performance. The cause of this was traced to disk I/O operations.

As we recall we did not see much improvement when we performed the scaling up, just a little, and since HDFS seems to affect the performance, we believe that HDFS is the reason we do not see a major improvement in the time of execution as we thought at first.

To verify that the current belief that HDFS affects the final time required by the cluster to finish the execution of the ongoing set of applications, we tried a test in which instead of scaling both HDFS and YARN at the same time, we only scale up YARN; done easily by just starting the YARN daemon but not the HDFS one. This means that only the 3 required slaves for HDFS will be running regardless of the amount of VMs we add to the cluster, this in theory will liberate the new VMs from HDFS related operations (read/write) and will freely deliver the resources available to YARN; expecting in this way a major reduction in time of execution. In this new test we used a 5 apps at a time each 10 minutes over the course of an hour test that we later escalated to up to 6 YARN slaves adding 1 node each 20 minutes; the results can be seen in the figure below.

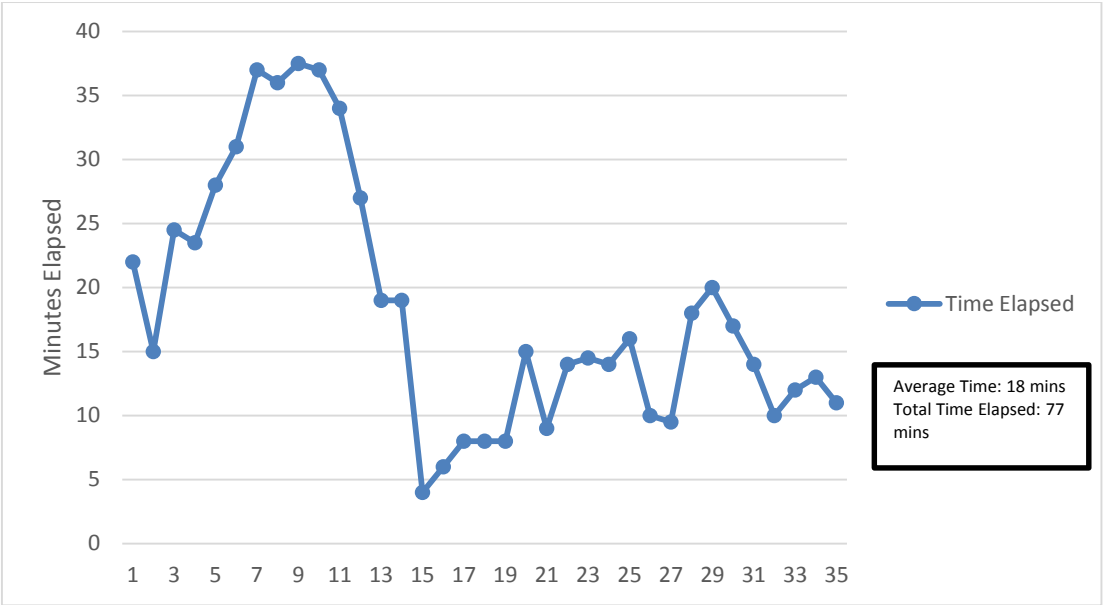

Figure 26: Elapsed, average and total time of 35 apps without HDFS

From the image below we can now see that without HDFS the times of execution of the apps are actually quite fast because we see and aggressive descend in the time of executions at the time of the first 20 minutes were we add the first new node and while it grows at some point the time of execution do not go beyond the 20 minutes mark, comparing them to our initial scaling tests results found in figure 21. The total time for the executions comes at only 77 minutes comparing it with the 113 minutes of the aforementioned test, with an average time of only 18 minutes compared to the 19 minutes of before. While there might not be too much of a difference between the average time of the test using HDFS and the test without the use of HDFS in the other 3 slaves, the total time elapsed was sufficiently smaller and the amount of failed applications were lesser in the former to consider other alternatives to HDFS in some cases; we will definitively take this finding into account for subsequent tests in the next phase.

As we just saw, many of the problems that we encountered were mainly caused by network issues but as mentioned we also found errors that are related to the I/O inputs of the disk, we believe that this is caused by one particular thing: we only used one server with just one specific hard drive for all the different roles in our cluster which seems to clearly be a bottleneck. This of course happens when we use a manual setting, more problems are set to arise when we start the next phase were everything should be done autonomously or semi-autonomously and more and more problems could arise when the cluster needs to be proactive.

While YARN and HDFS components are the most prominent troublemakers, we could track some errors caused due to the escalation process, especially when scaling up (since almost all of the scaling down was performed when idle). Hadoop will try its best to seamlessly add the new resources, at some point and we believe due to the fact of the one hard drive and as the results of the test shows, the time elapsed for the apps when adding resources were reduced but at some point it again went up, mainly because it will have more to do than just continue this execution, e.g. sending information to the new node. So this is also something to take into account when performing the next set of tests for the next phase.

At the end of this phase we have run the significant amount of 2,100+ applications and a significant array of tests including of course the reiterations of tests due to the different problems encountered

and commented in this phase, all of this has served as our knowledge base for the next phase which we will comment in brief in the following section.

5.2 Results for the Knowledge Phase

As we specified in the last chapter, the first test we devised to verify the automation of the cluster is that of the time it takes for the cluster to start with the default settings of 2 masters and 3 slaves. Using the provided tools by OpenNebula, namely OneFlow and OneGate, we created a multi-tier application as it is described in the documentation (http://docs.opennebula.org/4.12/) which is our base cluster (5 VMs, 2 Masters, 3 Workers), we managed to get everything running in between 10 and 15 minutes, it varied depending on networking connectivity since the frontend sends everything to our physical server where the VMs will be created. In the image seen below, we can look at the time in the log messages of OpenNebula corresponding to a test that took 10 minutes from the moment it started to deploy to the moment it was ready for execution, marked as RUNNING. In the same test we also took note of how much time it will take for the cluster to scale up, this will of course be approximate since there are other factors like the usage of the cluster in a given time, according to the log it only takes about 2 minutes to perform the escalation operation, however it takes about 8 minutes for everything to be again ready and running.

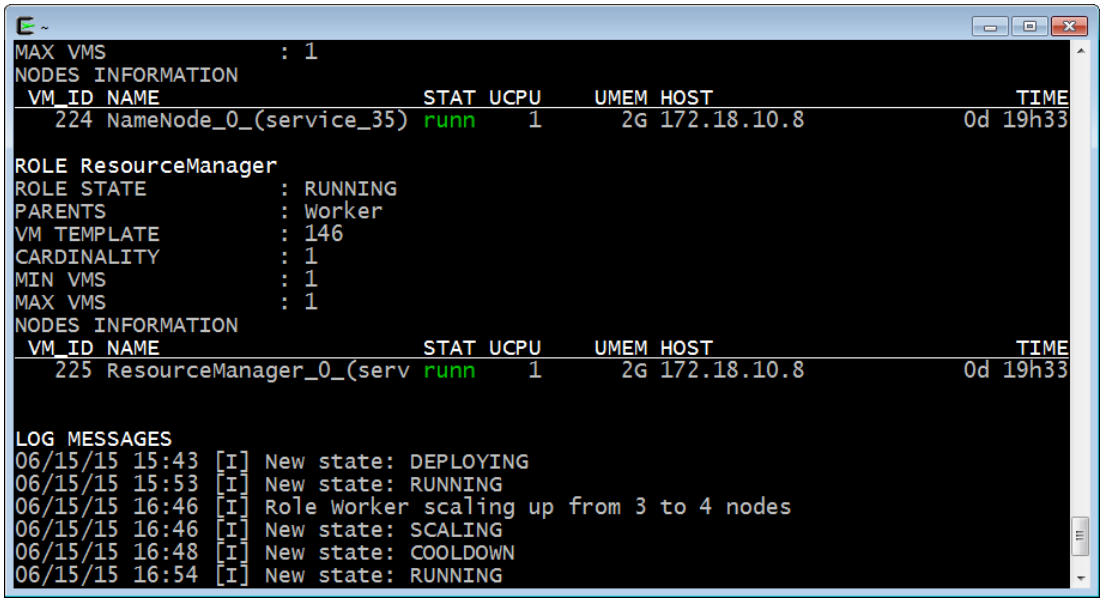

Figure 27: Log messages of Automation in OpenNebula

To be able to see if the final automation of this cluster is working as it is supposed to, we run a test of 5 applications with the same size to compare it with the first results we got from the understanding phase; this will be the validation we need to continue testing within this phase.

Figure 28: CPU load for 5 applications in the newly formed cluster

We can clearly asses from the image above that the cluster is fully functionally and that we can continue the testing process for this phase and see if we can make Hadoop autonomously elastic. Sadly and despite the many efforts performed to finish this phase, we were finally face with the reality that both the use case in hand as well as the required steps to test and validate the autonomic and proactive innovations of PANACEA in the given time was too ambitious. Therefore, we are unable to continue the tests devised for this phase even if the cluster seems ready, nevertheless we are sure that all the work performed during the understanding phase is enough to progress through this phase without problems in the future and we are looking forward to the completion of the PANACEA project using all of the information gathered.

5.3 Lessons Learned

A good set of lessons were learned through the tests performed and in general while doing all of the work that corresponds with this thesis. As with the majority of lessons learned through life, these lessons were learned due to different problems encountered while performing an activity, the testing in our case. Hadoop is a very useful technology but a complex one and it does not come with too many shortcuts. Let us make a review of the different lessons that were learned during this four months:

- We cannot and should not force the decommission of a node when it is stuck: A recurring problem with HDFS was faced when scaling down. However, there is no easy way to force or accelerate the process when it is stuck. We found that the easiest way is to shut down the HDFS component completely and then restarting it this process could take as less as ten minutes but of course this is not always possible. Other options like using the balancer should ease the transition but sometimes it does not, ultimately what was the easiest fix was to include the node being decommissioned again, wait till it was confirmed and then execute the decommission process again.
- As long as the minimally replicated blocks are 100% there should not be a problem: True, but if by any reason this is not at a 100% and we decide to shut down a node being decommissioned it could produce errors or inconsistencies in the data that will ultimately be harder to fix. Be patient.
- Hadoop is very configurable but you need to take care of what you configure: We tested a functionality for the ResourceManager to be able to recover from a restart, while the documentation specified that it was not completed as of version 2.6.0, one part of it was and we decided to include it. However this caused a lot of problems when instantiating a new ResourceManager killing the daemons of all YARN related daemons in the cluster.

- It is a good idea to configure the resources and the queues or scheduler beforehand: While the default options are very good and should not have any problems whatsoever, in a real case scenario we may want to devise a specific set of resources or a priority queue for specific purposes, this is all configurable and should be done without hesitation to be able to deliver complex SLAs.
- Finding the 'sweet spot' for resources in Hadoop is tricky: To ease this, we should define a set of SLAs clearly before to actually know when the resources should be added and how many are needed to deliver the results we expect.
- Adding resources too late will not improve the overall result: This should not be a surprise, but we at some point expected that when a new node is added the time of executions will be cut almost immediately; while this could happen with a 'big' enough VM, it was not our case and therefore should not be considered as something that will happen. Add resources when needed or face the consequences.
- Sending MapReduce applications from the same machines is counterproductive: Another one in the category of 'no surprise', at the beginning and with small tests we used to send the applications from the same machines that will process this, while at the beginning it was not a problem, once we added enough application to the queue, the problems started getting peak signatures and failing applications. We designated then a new VM that will be in charge of sending all of the applications.
- Sending too many applications from the same VM is also counterproductive: While the newly designated VM faced less problems than the case said before it was not exempt of problems as with enough applications (around the 50 apps) the problems with not enough memory started to arise; in a real case scenario one VM or user machine will not send a throng of applications as it will most than likely send one or two and wait for them to finish while other user will do just the same in another different machine, however it is good to know and measures like having enough resources in the designated VM should be taken. Do not overcharge the VMs in the cluster, or out of it.
- Different metrics require different review periods: When monitoring metrics we could be tempted to check everything every 10 seconds or less. However, some metrics need to be reviewed with more ease, take for example the HDFS space left, while it is a metrics that should be monitored it will not grow over 20 gigabytes over 10 seconds, it will take a little more than that and it will vary as we add or reduce VMs to our cluster. Checking metrics too soon could in fact add some overhead to the process thus hindering the performance of the cluster as a whole. Know the metrics and when you should review them.

CHAPTER 6 - CONCLUSSIONS AND FUTURE WORK

It has been quite a journey with this case of study from PANACEA; through it we can conclude a set of diverse things: starting from the fact that the complexity of testing a functional Cloud while looking for answers in a controlled environment that hopefully will help in the real case scenario is quite high.

We can honestly say that making Hadoop elastic is not an easy task; many things come into consideration and consequently there is no surprise that the current offering of this kind of characteristic for Hadoop is a little scarce. If making Hadoop elastic is hard work as it is for administrators, teaching the tasks to an algorithm to do it in a proactive and autonomous manner and not only that but doing it well is no short task, it is actually very challenging and ambitious.

Cloud infrastructures are quite complex mechanisms, and while our testing setting seemed to carry enough 'firepower' to tackle the concerns we wanted to dissipate about our use case for PANACEA, it seemed that it came a little too short and other settings should be done to relieve the problems encountered during the testing processes performed during this 4 months. While we actually took care of some points of view when designing the architecture, it was not until we were in the middle of testing that we actually found out about some short-sighted decisions we made along the way; but better during testing than in production, it is the nature of testing to find these problems and correct them before going into production.

While Hadoop is at some point a little simplistic, it was at some point quite hard to work with, it was however a satisfactory experience working with it. Although we did not were able to find out more about the problems that could arise during the life cycle of the use of the Hadoop framework, we firmly believe that all the information, experience and knowledge found in this document and in our minds will be most beneficial for future work both in industry or academia as well as in the PANACEA project as expected.

As we have said before, both PANACEA and the use case that was used in this work to validate the innovations intended by the project are very ambitious and as such there is substantial work to be done in the future. Therefore we will like to provide a future work agenda that anyone interested in the project could be able to follow:

- **Introduce testing with different sizes of virtual machines:** The testing we did was using very small virtual machines so we could scale up to a very large number of machines which in a real case scenario will undoubtedly happen. However, it will be interesting to perform the same tests using beefier machines to reduce the amount of virtual machines being hosted. As more resources are available for the workers, the workers should have enough power to outperform a cluster with smaller virtual machines.
- **Use the knowledge we gathered to set the base for the autonomic and proactive innovations of PANACEA:** We believe we gathered enough expertise to set the foundations for this innovation in the DAaaS use case, however there is a need for validation of this expertise as well as making it grow by using it.
- **Use other algorithms for testing:** A problem that we did not encounter, was memory leaks; as a Java program, MapReduce and Hadoop can suffer of this known problem. However the examples are simple enough that the probability of having memory leaks is very small. So we could use other more interesting algorithms mixed with some MapReduce examples to see how this will affect the cluster. In a real case scenario,

different users will have different needs as well as different programs so we need to be prepared for this scenario and teach the proactive and autonomic parts of PANACEA how to deal with memory leak problems as well as other problems that we may encounter when other technologies are being used in the cluster.

- **Use other storage options beside HDFS, or create an independent HDFS cluster altogether:** Most of the problems encountered during testing were coming from the HDFS part of the cluster. One interesting thing to test is using other technologies for storage besides HDFS based, for example Amazon EMR offers the use of S3 (its storage offering) that according to them can match the performance of HDFS. Other option could be to provide an independent cluster that runs HDFS specifically while another cluster performs the MapReduce and/or analytics part.
- **Look into the network:** Alongside HDFS, other problems that we found were caused by network issues (including HDFS related), e.g. the workers could not found each other at random, the workers could not reach the masters, could not replicate files across the cluster, etc. However we could not detect the specific cause for this issue since our monitoring technology could not give any hints and we can only speculate. It is true that part of what PANACEA aims to deliver includes being able to detect and repair network issues, but it would be very beneficial to look into the issues with other monitoring tools that could give a bigger picture of what the issues with the network are and what could be the cause of them.

Bibliography

[1] A. F. R. G. A. D. J. R. K. A. K. G. L. D. P. A. R. I. S. a. M. Z. Michael Armbrust, "Above the Clouds: A Berkeley View of Cloud Computing," California, 2009.

[2] M. S. Konstantinos GIANNAKOURIS, "eurostat statistics explained," November 2014. [Online]. Available: http://ec.europa.eu/eurostat/statistics-explained/index.php/Cloud_computing_-_statistics_on_the_use_by_enterprises. [Accessed May 2015].

[3] S. G. H. &. L. S. T. Ghemawat, "The Google file system," *ACM SIGOPS operating systems review,* vol. 37, no. 5, pp. 29-43, 2003.

[4] J. D. a. S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, 2004.

[5] Accenture Technology Labs, "Hadoop Deployment Comparison Study," 2013.

[6] H. J. L. L. L. Q. S. W. a. X. S. Shadi Ibrahim, "Evaluating MapReduce on Virtual Machines: The Hadoop Case," in *First International Conference, CloudCom*, Beijing, China, 2009.

[7] PANACEA Consortium, "Panacea Home Page," [Online]. Available: http://projects.laas.fr/panacea-cloud/.

[8] S. K. R. R. Nikolas Roman Herbst, "Elasticity in Cloud Computing: What it is, and what it is not," in *International Conference on Autonomic Computing*, San Jose, CA, 2013.

[9] Amazon, "Amazon Web Services Elastic Map Reduce," Amazon Inc., 2015. [Online]. Available: http://aws.amazon.com/elasticmapreduce/.

[10] E. Collins, "Big Data in the Public Cloud," *IEEE Computer Society,* vol. 1, no. 2, pp. 13-15, 2014.

[11] K. K. a. C. M. Pierre Riteau, "Bringing Elastic MapReduce to Scientific Clouds," in *3rd Annual Workshop on Cloud Computing and Its Applications: Poster Session*, Argonne, Illinois, 2011.

[12] OpenStack, "OpenStack HomePage," 2015. [Online]. Available: https://www.openstack.org/.

[13] OpenStack, "Sahara Project 0.3," [Online]. Available: https://savanna.readthedocs.org/en/0.3/.

[14] R. KUMAR and B. B. PARASHAR, "Dynamic resource allocation and management using openstack," *Nova,* vol. 1, p. 21, 2010.

[15] A. Thaha, M. Singh, A. Amin, N. Ahmad and S. Kannan, "Hadoop in OpenStack: Data-location-aware cluster provisioning," in *4th World Congress on Information and Communication Technologies*, 2014.

[16] Cloudera Inc, "Cloudera HomePage," [Online]. Available: http://www.cloudera.com/content/cloudera/en/home.html.

[17] Cloudera Inc, "Cloudera Documentation Console," [Online]. Available: http://www.cloudera.com/content/cloudera-content/cloudera-docs/CM4Ent/4.7.0/Cloudera-Manager-Introduction/Cloudera-Manager-Introduction.html.

[18] P. L. L. P. D. H. Philip Zeyliger, "Centralized configuration and monitoring of a distributed computing cluster". US Patent US20130204948 A1, 8 August 2013.

[19] Qubole Inc., "Qubole HomePage," [Online]. Available: http://www.qubole.com/.

[20] J. S. Sarma, "Qubole Blog," 17 June 2012. [Online]. Available: http://www.qubole.com/blog/product/industrys-first-auto-scaling-hadoop-clusters/. [Accessed 20 May 2015].

[21] D. Feinleib, "The Intersection of Big Data, Mobile and Cloud Computing," in *Big Data Bootcamp*, CA, United States, Apress, 2014, pp. 85-101.

[22] HortonWorks Inc., "Hortonworks Data Platform," [Online]. Available: http://hortonworks.com/hdp/. [Accessed 1 06 2015].

[23] HortonWorks Inc., "Partners," [Online]. Available: http://hortonworks.com/partner/rackspace/. [Accessed 01 06 2015].

[24] A. a. B. T. Stipic, "How cloud computing is (not) changing the way we do BI," in *MIPRO*, Opatija, 2012.

[25] U. R. A. a. R. R. Chandrasekhar, "A comparative study of enterprise and open source big data analytical tools," in *IEEE Conference on ICT*, Jeju Island, 2013.

[26] Google Inc., "Hadoop on Google Cloud Platform - Google Cloud Platform," [Online]. Available: https://cloud.google.com/hadoop/what-is-hadoop. [Accessed 01 June 2015].

[27] The Ganglia Project, "Ganglia Monitoring System Homepage," [Online]. Available: http://ganglia.sourceforge.net/.

[28] J. Liu, F. Liu and N. Ansari, "Monitoring and analyzing big traffic data of a large-scale cellular network with Hadoop," *Network, IEEE ,* vol. 28, no. 4, pp. 32-39, 2014.

[29] S. D. L. Z. Z. L. Wenguo Wei, "An Improved Ganglia-Like Clusters Monitoring System," in *Grid and Cooperative Computing*, Shangai, China, Springer Berlin Heidelberg, 2004, pp. 89-96.

[30] R. B. a. D. J. Patel, "Performance Analysis of A Grid Monitoring System - Ganglia," *International Journal of Emerging Technology and Advanced Engineering,* vol. 3, no. 8, pp. 362-365, 2013.

[31] Z. Z. a. Y. L. Mingli Wu, "Application research of Hadoop resource monitoring system based on Ganglia and Nagios," in *IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, China, 2013.

[32] Nagios Enterprises, "Nagios Homepage," [Online]. Available: http://www.nagios.org/.

[33] P. Z. a. R. Kodali, "Useful Splunk Apps," in *Big Data Analytics Using Splunk*, California, US, Apress, 2013, pp. 323-343.

[34] B. Lakhe, "Hadoop Metrics and Their Relevance to Security," in *Practical Hadoop Security*, IL, US, Apress, 2014, pp. 183-189.

[35] Splunk Inc., "Splunk HomePage," [Online]. Available: http://www.splunk.com/. [Accessed 01 June 2015].

[36] Splunk Inc., "Splunk App for HadoopOps," [Online]. Available: https://splunkbase.splunk.com/app/1173/. [Accessed 01 June 2015].

[37] S. S. N. S. S. Vidhya, "Comparative Analysis of Diverse Collection of Big Data Analytic Tools," *International Journal of Computer, Control, Quantum and Information Engineering ,* vol. 8, no. 9, pp. 1530-1536, 2014.

[38] Zabbix LLC, "ZABBIX Homepage," [Online]. Available: http://www.zabbix.com/. [Accessed 1 June 2015].

[39] T. V. V. a. T. R. Janpan, "A virtual machine consolidation framework for CloudStack platforms," in *International Conference on Information Networking (ICOIN)*, Phuket, 2014.

[40] M. C. M. a. P. M. Andreolini, "A Scalable Architecture for Real-Time Monitoring of Large Information Systems," in *Second Symposium on Network Cloud Computing and Applications (NCCA)*, London, 2012.

[41] M. S. T. a. A. T. Kuratkar, "Cloud Management with Open Source Tools," *International Journal of Advanced Research in Computer Science ,* vol. 4, no. 6, 2013.

[42] Apache Software Foundation, "Apache Ambari," 2015. [Online]. Available: http://ambari.apache.org/.

[43] S. W. a. M. Siddalingaiah, "Apache Ambari," in *Pro Apache Hadoop*, MD, US, Apress, 2014, pp. 399-401.

[44] S. S. A. H. B. a. S. S. S. S. Aravianth, "An Efficient HADOOP Frameworks SQOOP and Ambari for Big Data Processing," *International Journal for Innovative Research in Science & Technology,* vol. 1, no. 10, pp. 252-255, 2015.

[45] SEQUENCEIQ Inc., "SEQUENCEIQ Homepage," 2015. [Online]. Available:

http://sequenceiq.com/.

[46] Apache Softare Foundation, "Apache Slider Incubating Page," [Online]. Available: http://slider.incubator.apache.org/. [Accessed 1 June 2015].

[47] H. L. G. L. N. B. L. D. F. B. C. a. S. B. Herodotos Herodotou, "Starfish: A Self-tuning system for Big Data Analytics," in *5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*, Asilomar, California, 2011.

[48] Duke University, "Starfish Homepage," [Online]. Available: https://www.cs.duke.edu/starfish/index.html. [Accessed 10 June 2015].

[49] N. J. K. K. a. H. F. Allahbaksh M. Asadullahy, "A Data-Centric Heuristic for Hadoop Provisioning in the Cloud," in *COMPUTE "13*, Vellore, Tamil Nadu, India, 2013.

[50] Y. X. a. G. Z. Yuanshun Dai, "Self-Healing and Hybrid Diagnosis in Cloud Computing," in *First International Conference, CloudCom 2009*, Beijing, China, 2009.

[51] C. M. N. P. E. F. a. P. R. Anca Iordache, "Resilin: Elastic MapReduce over Multiple Clouds," in *IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, Delft, The Netherlands, 2013.

[52] S. Babu, "Towards Automatic Optimization of MapReduce Programs," in *SoCC'10*, Indianapolis, Indiana, USA, 2010.

[53] Apache Software Foundation, "Hadoop Documentation," [Online]. Available: http://hadoop.apache.org/docs/stable/. [Accessed 06 2015].

[54] OpenNebula Project, "OpenNebula HomePage," [Online]. Available: http://opennebula.org/.

[55] F. o. Forster, "Collectd Start Page," [Online]. Available: https://collectd.org/.

[56] Apache Software Foundation, "Hadoop Wiki," [Online]. Available: http://wiki.apache.org/hadoop/FrontPage. [Accessed 1 June 2015].

[57] G. B. A. Attebury, K. Bloom, B. Bockelman, D. Kcira, J. Letts, T. Levshina, C. Lundestedt, T. Martin, W. Maier, H. Pi, A. Rana, I. Sfiligoi, A. Sim, M. Thomas and F. Wuerthwein, "Hadoop distributed file system for the Grid," in *IEEE Nuclear Science Symposium Conference Record (NSS/MIC)*, Orlando, FL, 2009.

ANNEX 1

Testing the cluster before deploying in OpenNebula

We actually used this steps to run the first set of tests, the understanding phase, and is quite simple to perform. This step is also recommended to verify that everything works as it is intended and it will not have any trouble being run in an automated manner using, in this case, OpenNebula.
The very first step is to create one or several xmls to create the virtual machines for the KVM hypervisor, an example can be found below:

```
<domain type='kvm'>
 <name>Hadoop_Qcow</name>
 <memory unit='KiB'>2097152</memory>
 <currentMemory unit='KiB'>2097152</currentMemory>
 <vcpu placement='static'>2</vcpu>
 <resource>
  <partition>/machine</partition>
 </resource>
 <os>
  <type arch='x86_64' machine='pc-i440fx-trusty'>hvm</type>
  <boot dev='hd'/>
  <boot dev='cdrom'/>
 </os>
 <features>
  <acpi/>
 </features>
 <clock offset='utc'/>
 <on_poweroff>destroy</on_poweroff>
 <on_reboot>restart</on_reboot>
 <on_crash>restart</on_crash>
 <devices>
  <emulator>/usr/bin/kvm</emulator>
  <disk type='file' device='disk'>
   <driver name='qemu' type='qcow2'/>
   <source file='/home/atos/img_hadoop/hadoop_resource.qcow2'/>
   <target dev='vda' bus='virtio'/>
   <alias name='virtio-disk0'/>
   <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
  </disk>
  <disk type='file' device='cdrom'>
   <driver name='qemu' type='raw'/>
   <source file='/home/atos/isos/ubuntu-14.04.2-server-amd64.iso'/>
   <target dev='hdc' bus='ide' tray='open'/>
   <readonly/>
   <alias name='ide0-1-0'/>
   <address type='drive' controller='0' bus='1' target='0' unit='0'/>
  </disk>
  <controller type='ide' index='0'>
   <alias name='ide0'/>
   <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
  </controller>
```

```
    <controller type='usb' index='0'>
      <alias name='usb0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x2'/>
    </controller>
    <controller type='pci' index='0' model='pci-root'>
      <alias name='pci.0'/>
    </controller>
    <interface type='network'>
      <source network='default'/>
    </interface>
    <serial type='pty'>
      <source path='/dev/pts/1'/>
      <target port='0'/>
      <alias name='serial0'/>
    </serial>
    <console type='pty' tty='/dev/pts/1'>
      <source path='/dev/pts/1'/>
      <target type='serial' port='0'/>
      <alias name='serial0'/>
    </console>
    <input type='mouse' bus='ps2'/>
    <input type='keyboard' bus='ps2'/>
    <graphics type='vnc' port='5900' autoport='yes' listen='172.18.10.8'>
      <listen type='address' address='172.18.10.8'/>
    </graphics>
    <video>
      <model type='cirrus' vram='9216' heads='1'/>
      <alias name='video0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
    </video>
    <memballoon model='virtio'>
      <alias name='balloon0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </memballoon>
  </devices>
  <seclabel type='dynamic' model='apparmor' relabel='yes'>
    <label>libvirt-887ee873-485a-4204-a5fd-90aac6571d09</label>
    <imagelabel>libvirt-887ee873-485a-4204-a5fd-90aac6571d09</imagelabel>
  </seclabel>
</domain>
```

Pay attention if you are using only one xml to create all of the virtual machines required for testing the cluster as you need to change both the name and the .qcow2 file intended to work as the hard drive of the virtual machine.

Once you have the xml or xmls prepared you can then execute the following instruction as much as required (at least 5, two masters and three slaves).

```
$: virsh create name_of_xml_file
```

Once all the domains are running (verifiable by issuing a *virsh list* command) you can enter to configure and execute the commands required for the cluster to be operative.

Since we have installed the OpenNebula package, the required ip configuration is deleted so entering via ssh is not possible at this point, therefore it is required to install virt-viewer in the KVM

hypervisor machine and enter one by one to each of the virtual machines created by using the following.

```
$: virt-viewer number_or_name_of_VM
```

This will open a window that will be the virtual machine, the user is hadoop and the password is 'hadoop1' after that we can start modifying, in short we need to change 2 different files in the masters and 3 in the slaves.

Before we do anything, we changed the hostnames of the different slaves from Hadoop to Hop-Worker-1, Hop-Worker-2 and Hop-Worker-3, respectively by changing the /etc/hostaname and after that issuing a reboot of the virtual machine to continue modifying it in the same form as the masters.

```
$: sudo nano /etc/hostname
$: sudo reboot
```

Now for the masters and the rest of slaves, we need to change the **/etc/hosts** so it will know the different routes to the different virtual machines that are inside the cluster and we need also to change the **/etc/network/interfaces** file because at the moment we do not possess network capability. First we decided on the /etc/network/interfaces so we have network capability and then we know which is which in the cluster and then the /etc/hosts file can be modified accordingly. The following is an example of how our network is deployed and only one of the machines, bear in mind that it will not reflect the reality for other machines so you will have to ask or change the ip addresses accordingly.

```
auto eth0
iface eth0 inet static
address 192.168.122.5
netmask 255.255.255.0
broadcast 192.168.122.255
```

Once you have modified such file, we will need to issue the following command so that the eth0 interface may be available.

```
:$ sudo ifup eth0
```

We can verify then by using ifconfig and checking if the eth0 interface is indeed listed and ready, we can also ping the same address of the interface.

Now that we know the different ip address of the VMs that conform the cluster, we can modify the /etc/hosts file, such file will have the following (addresses may vary of course)

```
192.168.122.5 Hop-NameNode
192.168.122.6 Hop-ResourceMan
192.168.122.7 Hop-Worker-1
192.168.122.8 Hop-Worker-2
192.168.122.9 Hop-Worker-3
```

Once everything is up and running in all of the VMs, we need to enter to the masters and modify one last file so that we can issue a command that will initiate the Hadoop cluster. The file we are talking about is the one found under /usr/loca/hadoop/hadoop-2.6.0/etc/hadoop/slaves this file can contain the ip address or the hostnames of the slaves (one per line) and as such it will have the following structure.

```
Hop-Worker-1
Hop-Worker-2
Hop-Worker-3
```

Once it has been modified, we can issue one command in each master that will communicate through ssh to the slaves and execute the different daemons intended to be used by the Hadoop cluster.

Hop-NameNode

```
$: hdfs namenode –format
$: start-dfs.sh
```

Hop-ResourceMan

```
$: start-yarn.sh
```

It can take as long as 10 minutes for everything to run from scratch but after that we can verify that the services are correctly running by executing **jps** in the terminal of the different machines, the output of this command will vary depending on which machine you execute it in. In general you will be seeing the following (numbers will change as they are random):

Hop-NameNode

```
3522 NameNode
6227 Jps
```

Hop-ResourceMan

```
3265 SecondaryNameNode
21857 Jps
21157 ResourceManager
```

Hop-Worker-*

```
9056 Jps
8935 NodeManager
2496 DataNode
```

If the configuration goes as planned and the services are running, we should be able to access http://hop-namenode:50070 (or the IP of the machine) and see 3 nodes under Live Nodes; this is the HDFS. We should also see 3 nodes at http://hop-resourceman:8088/cluster/nodes (or the IP of the machine); this is the YARN component.

To test that everything indeed is working run one of the examples provided by the version of Hadoop, preferably in one of the masters, by executing the following:

```
yarn jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.0.jar pi 32 10000
```

A job should appear under the applications section in the YARN component and when it finishes you should have the following output in the machine where the example was executed: **Estimated value of Pi is 3.14147500000000000000**.

ANNEX 2

Codes for metric collection

hadoop-yarn-cluster-metrics.sh

```
#!/bin/bash

HOSTNAME="Hop-ResourceMan"
INTERVAL=10
HADOOP_METRICS_FILE="/tmp/resourcemanager-metrics.out"
NODE_NAME="YarnCluster"


while sleep "$INTERVAL"
do

   LINE=`tail -n 100 $HADOOP_METRICS_FILE | grep yarn.ClusterMetrics | tail -n 1`
   TIME=`echo $LINE | cut -d" " -f 1`
   NumActiveNMs=`echo $LINE | awk '{ print $6 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   NumDecommissionedNMs=`echo $LINE | awk '{ print $7 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   NumLostNMs=`echo $LINE | awk '{ print $8 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   NumUnhealthyNMs=`echo $LINE | awk '{ print $9}' | cut -d"=" -f 2 | cut -d"," -f 1`
   NumRebootedNMs=`echo $LINE | awk '{ print $10}' | cut -d"=" -f 2 | cut -d"," -f 1`

   echo "PUTVAL $HOSTNAME/$NODE_NAME-NumActiveNMs/memory interval=$INTERVAL
$TIME:$NumActiveNMs"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-NumDecommissionedNMs/memory
interval=$INTERVAL $TIME:$NumDecommissionedNMs"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-NumLostNMs/memory interval=$INTERVAL
$TIME:$NumLostNMs"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-NumUnhealthyNMs/memory interval=$INTERVAL
$TIME:$NumUnhealthyNMs"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-NumRebootedNMs/memory interval=$INTERVAL
$TIME:$NumRebootedNMs"

done
```

hadoop-yarn-jvm-metrics.sh

```
#!/bin/bash

HOSTNAME="Hop-ResourceMan"
INTERVAL=10
HADOOP_METRICS_FILE="/tmp/resourcemanager-metrics.out"
NODE_NAME="RESOURCE-MANAGER-JVM"


while sleep "$INTERVAL"
do

LINE=`tail -n 100 $HADOOP_METRICS_FILE | grep jvm.JvmMetrics | tail -n 1`
```

```
TIME=`echo $LINE | cut -d" " -f 1`
MemHeapUsedM=`echo $LINE | awk '{ print $10 }' | cut -d"=" -f 2 | cut -d"," -f 1`
MemNonHeapUsedM=`echo $LINE | awk '{ print $7 }' | cut -d"=" -f 2 | cut -d"," -f 1`
MemNonHeapCommittedM=`echo $LINE | awk '{ print $8 }' | cut -d"=" -f 2 | cut -d"," -f 1`
MemNonHeapMaxM=`echo $LINE | awk '{ print $9 }' | cut -d"=" -f 2 | cut -d"," -f 1`
MemHeapCommittedM=`echo $LINE | awk '{ print $11 }' | cut -d"=" -f 2 | cut -d"," -f 1`
MemHeapMaxM=`echo $LINE | awk '{ print $12 }' | cut -d"=" -f 2 | cut -d"," -f 1`
MemMaxM=`echo $LINE | awk '{ print $13 }' | cut -d"=" -f 2 | cut -d"," -f 1`
GcCountPS_Scavenge=`echo $LINE | awk '{ print $15 }' | cut -d"=" -f 2 | cut -d"," -f 1`
GcTimeMillisPS_Scavenge=`echo $LINE | awk '{ print $17 }' | cut -d"=" -f 2 | cut -d"," -f 1`
GcCountPS_MarkSweep=`echo $LINE | awk '{ print $19 }' | cut -d"=" -f 2 | cut -d"," -f 1`
GcTimeMillisPS_MarkSweep=`echo $LINE | awk '{ print $21 }' | cut -d"=" -f 2 | cut -d"," -f 1`
GcCount=`echo $LINE | awk '{ print $22 }' | cut -d"=" -f 2 | cut -d"," -f 1`
GcTimeMillis=`echo $LINE | awk '{ print $23 }' | cut -d"=" -f 2 | cut -d"," -f 1`
ThreadsNew=`echo $LINE | awk '{ print $24 }' | cut -d"=" -f 2 | cut -d"," -f 1`
ThreadsRunnable=`echo $LINE | awk '{ print $25 }' | cut -d"=" -f 2 | cut -d"," -f 1`
ThreadsBlocked=`echo $LINE | awk '{ print $26 }' | cut -d"=" -f 2 | cut -d"," -f 1`
ThreadsWaiting=`echo $LINE | awk '{ print $27 }' | cut -d"=" -f 2 | cut -d"," -f 1`
ThreadsTimedWaiting=`echo $LINE | awk '{ print $28 }' | cut -d"=" -f 2 | cut -d"," -f 1`
ThreadsTerminated=`echo $LINE | awk '{ print $29 }' | cut -d"=" -f 2 | cut -d"," -f 1`

echo "PUTVAL $HOSTNAME/$NODE_NAME-MemHeapUsedM/memory interval=$INTERVAL
$TIME:$MemHeapUsedM"
echo "PUTVAL $HOSTNAME/$NODE_NAME-MemNonHeapUsedM/memory interval=$INTERVAL
$TIME:$MemNonHeapUsedM"
echo "PUTVAL $HOSTNAME/$NODE_NAME-MemNonHeapCommittedM/memory
interval=$INTERVAL $TIME:$MemNonHeapCommittedM"
echo "PUTVAL $HOSTNAME/$NODE_NAME-MemNonHeapMaxM/memory interval=$INTERVAL
$TIME:$MemNonHeapMaxM"
echo "PUTVAL $HOSTNAME/$NODE_NAME-MemHeapCommittedM/memory interval=$INTERVAL
$TIME:$MemHeapCommittedM"
echo "PUTVAL $HOSTNAME/$NODE_NAME-MemHeapMaxM/memory interval=$INTERVAL
$TIME:$MemHeapMaxM"
echo "PUTVAL $HOSTNAME/$NODE_NAME-MemMaxM/memory interval=$INTERVAL
$TIME:$MemMaxM"
echo "PUTVAL $HOSTNAME/$NODE_NAME-GcCountPS_Scavenge/memory interval=$INTERVAL
$TIME:$GcCountPS_Scavenge"
echo "PUTVAL $HOSTNAME/$NODE_NAME-GcTimeMillisPS_Scavenge/memory interval=$INTERVAL
$TIME:$GcTimeMillisPS_Scavenge"
echo "PUTVAL $HOSTNAME/$NODE_NAME-GcCountPS_MarkSweep/memory interval=$INTERVAL
$TIME:$GcCountPS_MarkSweep"
echo "PUTVAL $HOSTNAME/$NODE_NAME-GcTimeMillisPS_MarkSweep/memory
interval=$INTERVAL $TIME:$GcTimeMillisPS_MarkSweep"
echo "PUTVAL $HOSTNAME/$NODE_NAME-GcCount/memory interval=$INTERVAL
$TIME:$GcCount"
echo "PUTVAL $HOSTNAME/$NODE_NAME-GcTimeMillis/memory interval=$INTERVAL
$TIME:$GcTimeMillis"
echo "PUTVAL $HOSTNAME/$NODE_NAME-ThreadsNew/memory interval=$INTERVAL
$TIME:$ThreadsNew"
echo "PUTVAL $HOSTNAME/$NODE_NAME-ThreadsRunnable/memory interval=$INTERVAL
$TIME:$ThreadsRunnable"
```

```
echo "PUTVAL $HOSTNAME/$NODE_NAME-ThreadsBlocked/memory interval=$INTERVAL
$TIME:$ThreadsBlocked"
echo "PUTVAL $HOSTNAME/$NODE_NAME-ThreadsWaiting/memory interval=$INTERVAL
$TIME:$ThreadsWaiting"
echo "PUTVAL $HOSTNAME/$NODE_NAME-ThreadsTimedWaiting/memory interval=$INTERVAL
$TIME:$ThreadsTimedWaiting"
echo "PUTVAL $HOSTNAME/$NODE_NAME-ThreadsTerminated/memory interval=$INTERVAL
$TIME:$ThreadsTerminated"

done
```

hadoop-yarn-metrics-system.sh

```
#!/bin/bash

HOSTNAME="Hop-ResourceMan"
INTERVAL=10
HADOOP_METRICS_FILE="/tmp/resourcemanager-metrics.out"
NODE_NAME="METRICS_SYSTEM"


while sleep "$INTERVAL"
do

  LINE=`tail -n 100 $HADOOP_METRICS_FILE | grep "metricssystem.MetricsSystem:
Context=metricssystem" | tail -n 1`
  TIME=`echo $LINE | cut -d" " -f 1`
  NumActiveSources=`echo $LINE | awk '{ print $5 }' | cut -d"=" -f 2 | cut -d"," -f 1`
  NumAllSources=`echo $LINE | awk '{ print $6 }' | cut -d"=" -f 2 | cut -d"," -f 1`
  NumActiveSinks=`echo $LINE | awk '{ print $7 }' | cut -d"=" -f 2 | cut -d"," -f 1`
  NumAllSinks=`echo $LINE | awk '{ print $8}' | cut -d"=" -f 2 | cut -d"," -f 1`
  Sink_fileNumOps=`echo $LINE | awk '{ print $9}' | cut -d"=" -f 2 | cut -d"," -f 1`
  Sink_fileAvgTime=`echo $LINE | awk '{ print $10}' | cut -d"=" -f 2 | cut -d"," -f 1`
  Sink_fileDropped=`echo $LINE | awk '{ print $11}' | cut -d"=" -f 2 | cut -d"," -f 1`
  Sink_fileQsize=`echo $LINE | awk '{ print $12}' | cut -d"=" -f 2 | cut -d"," -f 1`
  SnapshotNumOps=`echo $LINE | awk '{ print $13}' | cut -d"=" -f 2 | cut -d"," -f 1`
  SnapshotAvgTime=`echo $LINE | awk '{ print $14}' | cut -d"=" -f 2 | cut -d"," -f 1`
  PublishNumOps=`echo $LINE | awk '{ print $15}' | cut -d"=" -f 2 | cut -d"," -f 1`
  PublishAvgTime=`echo $LINE | awk '{ print $15}' | cut -d"=" -f 2 | cut -d"," -f 1`
  DroppedPubAll=`echo $LINE | awk '{ print $15}' | cut -d"=" -f 2 | cut -d"," -f 1`

  echo "PUTVAL $HOSTNAME/$NODE_NAME-NumActiveSources/memory interval=$INTERVAL
$TIME:$NumActiveSources"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-NumAllSources/memory interval=$INTERVAL
$TIME:$NumAllSources"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-NumActiveSinks/memory interval=$INTERVAL
$TIME:$NumActiveSinks"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-NumAllSinks/memory interval=$INTERVAL
$TIME:$NumAllSinks"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-Sink_fileNumOps/memory interval=$INTERVAL
$TIME:$Sink_fileNumOps"
  echo "PUTVAL $HOSTNAME/$NODE_NAME-Sink_fileAvgTime/memory interval=$INTERVAL
```

```
$TIME:$Sink_fileAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-Sink_fileDropped/memory interval=$INTERVAL
$TIME:$Sink_fileDropped"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-Sink_fileQsize/memory interval=$INTERVAL
$TIME:$Sink_fileQsize"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-SnapshotNumOps/memory interval=$INTERVAL
$TIME:$SnapshotNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-SnapshotAvgTime/memory interval=$INTERVAL
$TIME:$SnapshotAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-PublishNumOps/memory interval=$INTERVAL
$TIME:$PublishNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-PublishAvgTime/memory interval=$INTERVAL
$TIME:$PublishAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-DroppedPubAll/memory interval=$INTERVAL
$TIME:$DroppedPubAll"

done
```

hadoop-yarn-queue-metrics-root.sh

```
#!/bin/bash

HOSTNAME="Hop-ResourceMan"
INTERVAL=10
HADOOP_METRICS_FILE="/tmp/resourcemanager-metrics.out"
NODE_NAME="YarnQueueMetricsRoot"


while sleep "$INTERVAL"
do

   LINE=`tail -n 100 $HADOOP_METRICS_FILE | grep yarn.ClusterMetrics | tail -n 1`
   TIME=`echo $LINE | cut -d" " -f 1`
   running_0=`echo $LINE | awk '{ print $6 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   running_60=`echo $LINE | awk '{ print $7 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   running_300=`echo $LINE | awk '{ print $8 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   running_1440=`echo $LINE | awk '{ print $9}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AppsSubmitted=`echo $LINE | awk '{ print $10}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AppsRunning=`echo $LINE | awk '{ print $11}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AppsPending=`echo $LINE | awk '{ print $12}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AppsCompleted=`echo $LINE | awk '{ print $13}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AppsKilled=`echo $LINE | awk '{ print $14}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AppsFailed=`echo $LINE | awk '{ print $15}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AllocatedMB=`echo $LINE | awk '{ print $16}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AllocatedVCores=`echo $LINE | awk '{ print $17}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AllocatedContainers=`echo $LINE | awk '{ print $18}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AggregateContainersAllocated=`echo $LINE | awk '{ print $19}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AggregateContainersReleased=`echo $LINE | awk '{ print $20}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AvailableMB=`echo $LINE | awk '{ print $21}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AvailableVCores=`echo $LINE | awk '{ print $22}' | cut -d"=" -f 2 | cut -d"," -f 1`
   PendingMB=`echo $LINE | awk '{ print $23}' | cut -d"=" -f 2 | cut -d"," -f 1`
   PendingVCores=`echo $LINE | awk '{ print $24}' | cut -d"=" -f 2 | cut -d"," -f 1`
```

```
    PendingContainers=`echo $LINE | awk '{ print $25}' | cut -d"=" -f 2 | cut -d"," -f 1`
    ReservedMB=`echo $LINE | awk '{ print $26}' | cut -d"=" -f 2 | cut -d"," -f 1`
    ReservedVCores=`echo $LINE | awk '{ print $27}' | cut -d"=" -f 2 | cut -d"," -f 1`
    ReservedContainers=`echo $LINE | awk '{ print $28}' | cut -d"=" -f 2 | cut -d"," -f 1`
    ActiveUsers=`echo $LINE | awk '{ print $29}' | cut -d"=" -f 2 | cut -d"," -f 1`
    ActiveApplications=`echo $LINE | awk '{ print $30}' | cut -d"=" -f 2 | cut -d"," -f 1`

    echo "PUTVAL $HOSTNAME/$NODE_NAME-running_0/memory interval=$INTERVAL
$TIME:$running_0"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-running_60/memory interval=$INTERVAL
$TIME:$running_60"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-running_300/memory interval=$INTERVAL
$TIME:$running_300"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-running_1440/memory interval=$INTERVAL
$TIME:$running_1440"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-AppsSubmitted/memory interval=$INTERVAL
$TIME:$AppsSubmitted"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-AppsRunning/memory interval=$INTERVAL
$TIME:$AppsRunning"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-AppsPending/memory interval=$INTERVAL
$TIME:$AppsPending"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-AppsCompleted/memory interval=$INTERVAL
$TIME:$AppsPending"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-AppsFailed/memory interval=$INTERVAL
$TIME:$AppsFailed"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-AppsKilled/memory interval=$INTERVAL
$TIME:$AppsKilled"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-AllocatedMB/memory interval=$INTERVAL
$TIME:$AllocatedMB"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-AllocatedVCores/memory interval=$INTERVAL
$TIME:$AllocatedVCores"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-AllocatedContainers/memory interval=$INTERVAL
$TIME:$AllocatedContainers"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-AggregateContainersAllocated/memory
interval=$INTERVAL $TIME:$AggregateContainersAllocated"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-AggregateContainersReleased/memory
interval=$INTERVAL $TIME:$AggregateContainersReleased"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-AvailableMB/memory interval=$INTERVAL
$TIME:$AvailableMB"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-AvailableVCores/memory interval=$INTERVAL
$TIME:$AvailableVCores"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-PendingMB/memory interval=$INTERVAL
$TIME:$PendingMB"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-PendingVCores/memory interval=$INTERVAL
$TIME:$PendingVCores"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-PendingContainers/memory interval=$INTERVAL
$TIME:$PendingContainers"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-ReservedMB/memory interval=$INTERVAL
$TIME:$ReservedMB"
    echo "PUTVAL $HOSTNAME/$NODE_NAME-ReservedVCores/memory interval=$INTERVAL
$TIME:$ReservedVCores"
```

```
   echo "PUTVAL $HOSTNAME/$NODE_NAME-ReservedContainers/memory interval=$INTERVAL
$TIME:$ReservedContainers"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-ActiveUsers/memory interval=$INTERVAL
$TIME:$ActiveUsers"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-ActiveApplications/memory interval=$INTERVAL
$TIME:$ActiveApplications"

done
```

hadoop-yarn-queue-metrics-root-default.sh

```
#!/bin/bash

HOSTNAME="Hop-ResourceMan"
INTERVAL=10
HADOOP_METRICS_FILE="/tmp/resourcemanager-metrics.out"
NODE_NAME="YarnQueueMetricsRootDefault"


while sleep "$INTERVAL"
do

   LINE=`tail -n 100 $HADOOP_METRICS_FILE | grep "yarn.QueueMetrics: Queue=root.default" | tail
-n 1`
   TIME=`echo $LINE | cut -d" " -f 1`
   running_0=`echo $LINE | awk '{ print $6 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   running_60=`echo $LINE | awk '{ print $7 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   running_300=`echo $LINE | awk '{ print $8 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   running_1440=`echo $LINE | awk '{ print $9}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AppsSubmitted=`echo $LINE | awk '{ print $10}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AppsRunning=`echo $LINE | awk '{ print $11}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AppsPending=`echo $LINE | awk '{ print $12}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AppsCompleted=`echo $LINE | awk '{ print $13}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AppsKilled=`echo $LINE | awk '{ print $14}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AppsFailed=`echo $LINE | awk '{ print $15}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AllocatedMB=`echo $LINE | awk '{ print $16}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AllocatedVCores=`echo $LINE | awk '{ print $17}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AllocatedContainers=`echo $LINE | awk '{ print $18}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AggregateContainersAllocated=`echo $LINE | awk '{ print $19}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AggregateContainersReleased=`echo $LINE | awk '{ print $20}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AvailableMB=`echo $LINE | awk '{ print $21}' | cut -d"=" -f 2 | cut -d"," -f 1`
   AvailableVCores=`echo $LINE | awk '{ print $22}' | cut -d"=" -f 2 | cut -d"," -f 1`
   PendingMB=`echo $LINE | awk '{ print $23}' | cut -d"=" -f 2 | cut -d"," -f 1`
   PendingVCores=`echo $LINE | awk '{ print $24}' | cut -d"=" -f 2 | cut -d"," -f 1`
   PendingContainers=`echo $LINE | awk '{ print $25}' | cut -d"=" -f 2 | cut -d"," -f 1`
   ReservedMB=`echo $LINE | awk '{ print $26}' | cut -d"=" -f 2 | cut -d"," -f 1`
   ReservedVCores=`echo $LINE | awk '{ print $27}' | cut -d"=" -f 2 | cut -d"," -f 1`
   ReservedContainers=`echo $LINE | awk '{ print $28}' | cut -d"=" -f 2 | cut -d"," -f 1`
   ActiveUsers=`echo $LINE | awk '{ print $29}' | cut -d"=" -f 2 | cut -d"," -f 1`
   ActiveApplications=`echo $LINE | awk '{ print $30}' | cut -d"=" -f 2 | cut -d"," -f 1`

   echo "PUTVAL $HOSTNAME/$NODE_NAME-running_0/memory interval=$INTERVAL
```

```
$TIME:$running_0"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-running_60/memory interval=$INTERVAL
$TIME:$running_60"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-running_300/memory interval=$INTERVAL
$TIME:$running_300"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-running_1440/memory interval=$INTERVAL
$TIME:$running_1440"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AppsSubmitted/memory interval=$INTERVAL
$TIME:$AppsSubmitted"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AppsRunning/memory interval=$INTERVAL
$TIME:$AppsRunning"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AppsPending/memory interval=$INTERVAL
$TIME:$AppsPending"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AppsCompleted/memory interval=$INTERVAL
$TIME:$AppsPending"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AppsFailed/memory interval=$INTERVAL
$TIME:$AppsFailed"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AppsKilled/memory interval=$INTERVAL
$TIME:$AppsKilled"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AllocatedMB/memory interval=$INTERVAL
$TIME:$AllocatedMB"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AllocatedVCores/memory interval=$INTERVAL
$TIME:$AllocatedVCores"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AllocatedContainers/memory interval=$INTERVAL
$TIME:$AllocatedContainers"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AggregateContainersAllocated/memory
interval=$INTERVAL $TIME:$AggregateContainersAllocated"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AggregateContainersReleased/memory
interval=$INTERVAL $TIME:$AggregateContainersReleased"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AvailableMB/memory interval=$INTERVAL
$TIME:$AvailableMB"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AvailableVCores/memory interval=$INTERVAL
$TIME:$AvailableVCores"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-PendingMB/memory interval=$INTERVAL
$TIME:$PendingMB"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-PendingVCores/memory interval=$INTERVAL
$TIME:$PendingVCores"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-PendingContainers/memory interval=$INTERVAL
$TIME:$PendingContainers"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-ReservedMB/memory interval=$INTERVAL
$TIME:$ReservedMB"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-ReservedVCores/memory interval=$INTERVAL
$TIME:$ReservedVCores"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-ReservedContainers/memory interval=$INTERVAL
$TIME:$ReservedContainers"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-ActiveUsers/memory interval=$INTERVAL
$TIME:$ActiveUsers"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-ActiveApplications/memory interval=$INTERVAL
$TIME:$ActiveApplications"

done
```

hadoop-yarn-rpc-8025.sh

```
#!/bin/bash

HOSTNAME="Hop-ResourceMan"
INTERVAL=10
HADOOP_METRICS_FILE="/tmp/resourcemanager-metrics.out"
NODE_NAME="RPC-8025"


while sleep "$INTERVAL"
do

   LINE=`tail -n 100 $HADOOP_METRICS_FILE | grep "rpc.rpc: port=8025" | tail -n 1`
   TIME=`echo $LINE | cut -d" " -f 1`
   ReceivedBytes=`echo $LINE | awk '{ print $6 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   SentBytes=`echo $LINE | awk '{ print $7 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcQueueTimeNumOps=`echo $LINE | awk '{ print $8 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcQueueTimeAvgTime=`echo $LINE | awk '{ print $9}' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcProcessingTimeNumOps=`echo $LINE | awk '{ print $10}' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcProcessingTimeAvgTime=`echo $LINE | awk '{ print $11}' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcAuthenticationFailures=`echo $LINE | awk '{ print $12}' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcAuthenticationSuccesses=`echo $LINE | awk '{ print $13}' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcAuthorizationFailures=`echo $LINE | awk '{ print $14}' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcAuthorizationSuccesses=`echo $LINE | awk '{ print $15}' | cut -d"=" -f 2 | cut -d"," -f 1`
   NumOpenConnections=`echo $LINE | awk '{ print $16}' | cut -d"=" -f 2 | cut -d"," -f 1`
   CallQueueLength=`echo $LINE | awk '{ print $17}' | cut -d"=" -f 2 | cut -d"," -f 1`

   echo "PUTVAL $HOSTNAME/$NODE_NAME-ReceivedBytes/memory interval=$INTERVAL
$TIME:$ReceivedBytes"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-SentBytes/memory interval=$INTERVAL
$TIME:$SentBytes"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcQueueTimeNumOps/memory interval=$INTERVAL
$TIME:$RpcQueueTimeNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcQueueTimeAvgTime/memory interval=$INTERVAL
$TIME:$RpcQueueTimeAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcProcessingTimeNumOps/memory
interval=$INTERVAL $TIME:$RpcProcessingTimeNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcProcessingTimeAvgTime/memory
interval=$INTERVAL $TIME:$RpcProcessingTimeAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcAuthenticationFailures/memory
interval=$INTERVAL $TIME:$RpcAuthenticationFailures"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcAuthenticationSuccesses/memory
interval=$INTERVAL $TIME:$RpcAuthenticationSuccesses"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcAuthorizationFailures/memory
interval=$INTERVAL $TIME:$RpcAuthorizationFailures"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcAuthorizationSuccesses/memory
interval=$INTERVAL $TIME:$RpcAuthorizationSuccesses"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-NumOpenConnections/memory interval=$INTERVAL
$TIME:$NumOpenConnections"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-CallQueueLength/memory interval=$INTERVAL
```

```
$TIME:$CallQueueLength"

done
```

hadoop-yarn-rpc-8030.sh

```
#!/bin/bash

HOSTNAME="Hop-ResourceMan"
INTERVAL=10
HADOOP_METRICS_FILE="/tmp/resourcemanager-metrics.out"
NODE_NAME="RPC-8030"


while sleep "$INTERVAL"
do

   LINE=`tail -n 100 $HADOOP_METRICS_FILE | grep "rpc.rpc: port=8030" | tail -n 1`
   TIME=`echo $LINE | cut -d" " -f 1`
   ReceivedBytes=`echo $LINE | awk '{ print $6 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   SentBytes=`echo $LINE | awk '{ print $7 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcQueueTimeNumOps=`echo $LINE | awk '{ print $8 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcQueueTimeAvgTime=`echo $LINE | awk '{ print $9}' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcProcessingTimeNumOps=`echo $LINE | awk '{ print $10}' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcProcessingTimeAvgTime=`echo $LINE | awk '{ print $11}' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcAuthenticationFailures=`echo $LINE | awk '{ print $12}' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcAuthenticationSuccesses=`echo $LINE | awk '{ print $13}' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcAuthorizationFailures=`echo $LINE | awk '{ print $14}' | cut -d"=" -f 2 | cut -d"," -f 1`
   RpcAuthorizationSuccesses=`echo $LINE | awk '{ print $15}' | cut -d"=" -f 2 | cut -d"," -f 1`
   NumOpenConnections=`echo $LINE | awk '{ print $16}' | cut -d"=" -f 2 | cut -d"," -f 1`
   CallQueueLength=`echo $LINE | awk '{ print $17}' | cut -d"=" -f 2 | cut -d"," -f 1`

   echo "PUTVAL $HOSTNAME/$NODE_NAME-ReceivedBytes/memory interval=$INTERVAL
$TIME:$ReceivedBytes"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-SentBytes/memory interval=$INTERVAL
$TIME:$SentBytes"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcQueueTimeNumOps/memory interval=$INTERVAL
$TIME:$RpcQueueTimeNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcQueueTimeAvgTime/memory interval=$INTERVAL
$TIME:$RpcQueueTimeAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcProcessingTimeNumOps/memory
interval=$INTERVAL $TIME:$RpcProcessingTimeNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcProcessingTimeAvgTime/memory
interval=$INTERVAL $TIME:$RpcProcessingTimeAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcAuthenticationFailures/memory
interval=$INTERVAL $TIME:$RpcAuthenticationFailures"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcAuthenticationSuccesses/memory
interval=$INTERVAL $TIME:$RpcAuthenticationSuccesses"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcAuthorizationFailures/memory
interval=$INTERVAL $TIME:$RpcAuthorizationFailures"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RpcAuthorizationSuccesses/memory
interval=$INTERVAL $TIME:$RpcAuthorizationSuccesses"
```

```
   echo "PUTVAL $HOSTNAME/$NODE_NAME-NumOpenConnections/memory interval=$INTERVAL
$TIME:$NumOpenConnections"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-CallQueueLength/memory interval=$INTERVAL
$TIME:$CallQueueLength"

done
```

hadoop-yarn-rpc-detailed-port-8025.sh

```
#!/bin/bash

HOSTNAME="Hop-ResourceMan"
INTERVAL=10
HADOOP_METRICS_FILE="/tmp/resourcemanager-metrics.out"
NODE_NAME="RPC-DETAILED-8025"


while sleep "$INTERVAL"
do

   LINE=`tail -n 100 $HADOOP_METRICS_FILE | grep "rpcdetailed.rpcdetailed: port=8025" | tail -n 1`
   TIME=`echo $LINE | cut -d" " -f 1`
   RegisterApplicationMasterNumOps=`echo $LINE | awk '{ print $6 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   RegisterApplicationMasterAvgTime=`echo $LINE | awk '{ print $7 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   NodeHeartbeatNumOps=`echo $LINE | awk '{ print $8 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   NodeHeartbeatAvgTime=`echo $LINE | awk '{ print $9}' | cut -d"=" -f 2 | cut -d"," -f 1`

   echo "PUTVAL $HOSTNAME/$NODE_NAME-RegisterApplicationMasterNumOps/memory
interval=$INTERVAL $TIME:$RegisterApplicationMasterNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RegisterApplicationMasterAvgTime/memory
interval=$INTERVAL $TIME:$RegisterApplicationMasterAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-NodeHeartbeatNumOps/memory interval=$INTERVAL
$TIME:$NodeHeartbeatNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-NodeHeartbeatAvgTime/memory interval=$INTERVAL
$TIME:$NodeHeartbeatAvgTime"

done
```

hadoop-yarn-rpc-detailed-port-8030.sh

```
#!/bin/bash

HOSTNAME="Hop-ResourceMan"
INTERVAL=10
HADOOP_METRICS_FILE="/tmp/resourcemanager-metrics.out"
NODE_NAME="RPC-DETAILED-8030"


while sleep "$INTERVAL"
do

   LINE=`tail -n 100 $HADOOP_METRICS_FILE | grep "rpcdetailed.rpcdetailed: port=8030" | tail -n 1`
   TIME=`echo $LINE | cut -d" " -f 1`
```

```
   RegisterApplicationMasterNumOps=`echo $LINE | awk '{ print $6 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   RegisterApplicationMasterAvgTime=`echo $LINE | awk '{ print $7 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   AllocateNumOps=`echo $LINE | awk '{ print $8 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   AllocateAvgTime=`echo $LINE | awk '{ print $9}' | cut -d"=" -f 2 | cut -d"," -f 1`
   FinishApplicationMasterNumOps=`echo $LINE | awk '{ print $10}' | cut -d"=" -f 2 | cut -d"," -f 1`
   FinishApplicationMasterAvgTime=`echo $LINE | awk '{ print $11}' | cut -d"=" -f 2 | cut -d"," -f 1`

   echo "PUTVAL $HOSTNAME/$NODE_NAME-RegisterApplicationMasterNumOps/memory
interval=$INTERVAL $TIME:$RegisterApplicationMasterNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-RegisterApplicationMasterAvgTime/memory
interval=$INTERVAL $TIME:$RegisterApplicationMasterAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AllocateNumOps/memory interval=$INTERVAL
$TIME:$AllocateNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-AllocateAvgTime/memory interval=$INTERVAL
$TIME:$AllocateAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-FinishApplicationMasterNumOps/memory
interval=$INTERVAL $TIME:$FinishApplicationMasterNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-FinishApplicationMasterAvgTime/memory
interval=$INTERVAL $TIME:$FinishApplicationMasterAvgTime"

done
```

hadoop-yarn-rpc-detailed-port-8050

```
#!/bin/bash

HOSTNAME="Hop-ResourceMan"
INTERVAL=10
HADOOP_METRICS_FILE="/tmp/resourcemanager-metrics.out"
NODE_NAME="RPC-DETAILED-8050"


while sleep "$INTERVAL"
do

   LINE=`tail -n 100 $HADOOP_METRICS_FILE | grep "rpcdetailed.rpcdetailed: port=8050" | tail -n 1`
   TIME=`echo $LINE | cut -d" " -f 1`
   GetApplicationsNumOps=`echo $LINE | awk '{ print $6 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetApplicationsAvgTime=`echo $LINE | awk '{ print $7 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetNewApplicationNumOps=`echo $LINE | awk '{ print $8 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetNewApplicationAvgTime=`echo $LINE | awk '{ print $9}' | cut -d"=" -f 2 | cut -d"," -f 1`
   SubmitApplicationNumOps=`echo $LINE | awk '{ print $10}' | cut -d"=" -f 2 | cut -d"," -f 1`
   SubmitApplicationAvgTime=`echo $LINE | awk '{ print $11}' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetApplicationReportNumOps=`echo $LINE | awk '{ print $12}' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetApplicationReportAvgTime=`echo $LINE | awk '{ print $13}' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetApplicationAttemptsNumOps=`echo $LINE | awk '{ print $14}' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetApplicationAttemptsAvgTime=`echo $LINE | awk '{ print $15}' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetApplicationAttemptReportNumOps=`echo $LINE | awk '{ print $16}' | cut -d"=" -f 2 | cut -d"," -
f 1`
   GetApplicationAttemptReportAvgTime=`echo $LINE | awk '{ print $17}' | cut -d"=" -f 2 | cut -d"," -
f 1`
   GetContainersNumOps=`echo $LINE | awk '{ print $18}' | cut -d"=" -f 2 | cut -d"," -f 1`
```

```
   GetContainersAvgTime=`echo $LINE | awk '{ print $19}' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetContainerReportNumOps=`echo $LINE | awk '{ print $20}' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetContainerReportAvgTime=`echo $LINE | awk '{ print $21}' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetQueueInfoNumOps=`echo $LINE | awk '{ print $22}' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetQueueInfoAvgTime=`echo $LINE | awk '{ print $23}' | cut -d"=" -f 2 | cut -d"," -f 1`

   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetApplicationsNumOps/memory
interval=$INTERVAL $TIME:$GetApplicationsNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetApplicationsAvgTime/memory interval=$INTERVAL
$TIME:$GetApplicationsAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetNewApplicationNumOps/memory
interval=$INTERVAL $TIME:$GetNewApplicationNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetNewApplicationAvgTime/memory
interval=$INTERVAL $TIME:$GetNewApplicationAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-SubmitApplicationNumOps/memory
interval=$INTERVAL $TIME:$SubmitApplicationNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-SubmitApplicationAvgTime/memory
interval=$INTERVAL $TIME:$SubmitApplicationAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetApplicationReportNumOps/memory
interval=$INTERVAL $TIME:$GetApplicationReportNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetApplicationReportAvgTime/memory
interval=$INTERVAL $TIME:$GetApplicationReportAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetApplicationAttemptsNumOps/memory
interval=$INTERVAL $TIME:$GetApplicationAttemptsNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetApplicationAttemptsAvgTime/memory
interval=$INTERVAL $TIME:$GetApplicationAttemptsAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetApplicationAttemptReportNumOps/memory
interval=$INTERVAL $TIME:$GetApplicationAttemptReportNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetApplicationAttemptReportAvgTime/memory
interval=$INTERVAL $TIME:$GetApplicationAttemptReportAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetContainersNumOps/memory interval=$INTERVAL
$TIME:$GetContainersNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetContainersAvgTime/memory interval=$INTERVAL
$TIME:$GetContainersAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetContainerReportAvgTime/memory
interval=$INTERVAL $TIME:$GetContainerReportAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetQueueInfoNumOps/memory interval=$INTERVAL
$TIME:$GetQueueInfoNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetQueueInfoAvgTime/memory interval=$INTERVAL
$TIME:$GetQueueInfoAvgTime"

done
```

hadoop-yarn-ugi-metrics.sh

```
#!/bin/bash

HOSTNAME="Hop-ResourceMan"
INTERVAL=10
HADOOP_METRICS_FILE="/tmp/resourcemanager-metrics.out"
NODE_NAME="UGI"
```

```
while sleep "$INTERVAL"
do

   LINE=`tail -n 100 $HADOOP_METRICS_FILE | grep "ugi.UgiMetrics: Context=ugi" | tail -n 1`
   TIME=`echo $LINE | cut -d" " -f 1`
   LoginSuccessNumOps=`echo $LINE | awk '{ print $5 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   LoginSuccessAvgTime=`echo $LINE | awk '{ print $6 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   LoginFailureNumOps=`echo $LINE | awk '{ print $7 }' | cut -d"=" -f 2 | cut -d"," -f 1`
   LoginFailureAvgTime=`echo $LINE | awk '{ print $8}' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetGroupsNumOps=`echo $LINE | awk '{ print $9}' | cut -d"=" -f 2 | cut -d"," -f 1`
   GetGroupsAvgTime=`echo $LINE | awk '{ print $10}' | cut -d"=" -f 2 | cut -d"," -f 1`

   echo "PUTVAL $HOSTNAME/$NODE_NAME-LoginSuccessNumOps/memory interval=$INTERVAL
$TIME:$LoginSuccessNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-LoginSuccessAvgTime/memory interval=$INTERVAL
$TIME:$LoginSuccessAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-LoginFailureNumOps/memory interval=$INTERVAL
$TIME:$LoginFailureNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-LoginFailureAvgTime/memory interval=$INTERVAL
$TIME:$LoginFailureAvgTime"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetGroupsNumOps/memory interval=$INTERVAL
$TIME:$GetGroupsNumOps"
   echo "PUTVAL $HOSTNAME/$NODE_NAME-GetGroupsAvgTime/memory interval=$INTERVAL
$TIME:$GetGroupsAvgTime"

done
```

yarn-node-metrics.sh

```
#!/bin/bash

INTERVAL=60

while sleep "$INTERVAL"
do

 IFS=$'\r\n' GLOBIGNORE='*' :; NODES=($(yarn node --list 2> /dev/null | sed -e '1,/Node-Id/d' | awk
'{ print $1 }' ))

 for node in "${NODES[@]}"
 do

   NODE_NAME=`echo $node | cut -d ":" -f 1 `

   STATUS=`yarn node --status $node 2> /dev/null`

   CONTAINERS=`echo $STATUS | awk '{ print $25 }'`

   MEMORY_USED=`echo $STATUS | awk '{ print $28 }'`
   MEMORY_USED=${MEMORY_USED::-2}
```

```
   TOTAL_MEMORY=`echo $STATUS | awk '{ print $31 }'`
   TOTAL_MEMORY=${TOTAL_MEMORY::-2}

   CPU_USED=`echo $STATUS | awk '{ print $34 }'`

   CPU_CAPACITY=`echo $STATUS | awk '{ print $38 }'`

   DATE=`date +%s`

   echo "PUTVAL $NODE_NAME/YARN-Containers/memory interval=$INTERVAL
$DATE:$CONTAINERS"
   echo "PUTVAL $NODE_NAME/YARN-MemoryUsed/memory interval=$INTERVAL
$DATE:$MEMORY_USED"
   echo "PUTVAL $NODE_NAME/YARN-MemoryCapacity/memory interval=$INTERVAL
$DATE:$TOTAL_MEMORY"
   echo "PUTVAL $NODE_NAME/YARN-CPUUsed/memory interval=$INTERVAL $DATE:$CPU_USED"
   echo "PUTVAL $NODE_NAME/YARN-CPUCapacity/memory interval=$INTERVAL
$DATE:$CPU_CAPACITY"

  done
done
```

ANNEX 3

In this annex we define some of the bash scripts that we used for the automation of the cluster for the first time it enters.

In the following block we define the bash script that launches the cluster for the first time that uses the json that we mentioned in the chapter dedicated to it with some parameters given by the user. At the beginning we set up the ground for the different parts of the cluster, we deploy the images, create the templates and update the parameters accordingly in the json to later execute it and wait until it has been deployed.

```
#!/usr/bin/env ruby

LOCK = "lock"
$ip_init = 0
$ip_end = 0
ips_worker = []
hostnames_worker = []

#*************************************
# Input parameter onde inited the script *
#*************************************

#Ask for main path were the resources are

print "Enter PATH where the resources are : "
path = gets.chomp
puts "The path introduced is  #{path}"

print "Enter PATH where the images are : "
path_img = gets.chomp
puts "The path introduced is  #{path_img}"

#Ask for how many worker do we need

print "Enter the NUMBER of WORKERS NODES in the cluster : "
num_workers = gets.chomp
puts "The number of workers wished are #{num_workers}"

#=====================================================================
# Images with Ubuntu Server, contextualization packages and so on...
#=====================================================================
#Image for Hadoop Resource Manager: hadoop_resource.qcow2
#Image for Hadoop NameNode: hadoop_namenode.qcow2
#Image for Hadoop Worker: hadoop_worker.qcow2

#=====================================
# General functions used
#=====================================

#-----------------------
# Update templates files
#-----------------------

def changeOnFile(file, regex_to_find, text_to_put_in_place)
  text = File.read file
  File.open(file, 'w+'){|f| f << text.gsub(regex_to_find, text_to_put_in_place)}
end

#-----------------------------------
# Wait until images are ready to use
#-----------------------------------

def wait(id_image)
  puts "Waiting for the image STATE to be READY ...."
  output = %x[oneimage show "#{id_image}"]
  while output.include? "#{LOCK}" do
      # Wait for a while until he image is ready
      output = %x[oneimage show "#{id_image}"]
  end
end
```

```
#------------------------
# Get ID resource by Name
#------------------------

def getIDbyName(name)
  id = %x[oneimage show "#{name}"]
  id = id.scan(/(ID             : .*)/).last.first
  id = id.gsub!(/\D/, "")
  return id
end

#----------------------------
#Function to gets just number
#----------------------------

def getID(var)
   return var.delete('^0-9')
end

#-----------------------------------------------------
# Assign Initial IP and End IP of the private network
#-----------------------------------------------------

def assignRangeIPs(id_net)
  text = %x[onevnet show "#{id_net}"]
  $ip_init    =    text.match(/IP\s*([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)\s*([0-9]+\.[0-
9]+\.[0-9]+\.[0-9]+)/)[1]
  $ip_end     =    text.match(/IP\s*([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)\s*([0-9]+\.[0-
9]+\.[0-9]+\.[0-9]+)/)[2]
end

#--------------------------------------
#Get last ID of a VM (Not used finally)
#--------------------------------------

def getlastIDofVM()
  text = %x[onevm list]
  text = text.split("\n")
  text = text[text.length - 1]
  return text.match(/(\d\d)/)[1].to_i
end

#************************************************
#               Start all deployment            *
#************************************************

puts "Starting process of contextualization and deployment of Hadoop Cluster on
OpenNebula ..."

#================================
#Update PATH templates for images
#================================

changeOnFile("#{path}/cluster/templates/images/panacea_resourcemanager",
/..hadoop_resource.qcow2/, "#{path_img}/images/hadoop_resource.qcow2")

changeOnFile("#{path}/cluster/templates/images/panacea_namenode",
/..hadoop_namenode.qcow2/, "#{path_img}/images/hadoop_namenode.qcow2")

changeOnFile("#{path}/cluster/templates/images/panacea_worker",
/..hadoop_worker.qcow2/, "#{path_img}/images/hadoop_worker.qcow2")

puts "Updated PATH=#{path_img} for images templates"

#==========================
#Create datastore for images
#==========================

id_datastore                =                %x[onedatastore                create
"#{path}/cluster/templates/datastores/panacea_datastore"]
id_datastore = id_datastore.delete('^0-9')
puts "Datastore created with ID #{id_datastore}"

#=============
#Create images
#=============
```

```
sleep(4)

id_image_resourcemanager              =              %x[oneimage              create
"#{path}/cluster/templates/images/panacea_resourcemanager" -d "#{id_datastore}"]
id_image_resourcemanager = id_image_resourcemanager.delete('^0-9')

id_image_namenode                =                %x[oneimage                create
"#{path}/cluster/templates/images/panacea_namenode" -d "#{id_datastore}"]
id_image_namenode = id_image_namenode.delete('^0-9')

id_image_worker                  =                %x[oneimage                create
"#{path}/cluster/templates/images/panacea_worker" -d "#{id_datastore}"]
id_image_worker = id_image_worker.delete('^0-9')

puts "OS images created, Resource Manager OS image ID #{id_image_resourcemanager},
Namenode   OS   image   ID   #{id_image_namenode}   and   Worker   OS   image   ID
#{id_image_worker}"

#======================
#Create private network
#======================

id_network                    =                    %x[onevnet                    create
"#{path}/cluster/templates/networks/panacea_cluster_network"]
id_network = id_network.delete('^0-9')
puts "Private network created with ID #{id_network}"

#=================================================================
#Core process to set up the cluster network once init every machine
#=================================================================

assignRangeIPs(id_network)

segments = $ip_init.split('.')
int = $ip_init.split('.')[3].to_i
max = ($ip_end.split('.')[3].to_i -  $ip_init.split('.')[3].to_i) - 1

puts "The max workers according with the private network defined are #{max}"
puts "Checking whether #{num_workers} wished workers can be reach, otherwise will
set the above max value the number of workers in the cluster"

num_workers = num_workers.to_i

if num_workers > max
   puts  "Setting  to  #{max}  because  you  entered  a  wrong  number  of  workers
(#{num_workers}) according with your network configuration ..."
   num_workers = max
end

for i in 1..num_workers do
  ips_worker.push("#{segments[0]}.#{segments[1]}.#{segments[2]}.#{int}")
  int += 1
end

for i in 1..num_workers do
  hostnames_worker.push("Hop-Worker-#{i}")
  i += 1
end

ip_namenode = "#{segments[0]}.#{segments[1]}.#{segments[2]}.#{int}"
int += 1
ip_resource_manager = "#{segments[0]}.#{segments[1]}.#{segments[2]}.#{int}"

changeOnFile("#{path}/cluster/scripts/resourcemanager.sh",
/CLUSTER_IP_INIT=CLUSTER_IP_INIT\nCLUSTER_IP_END=CLUSTER_IP_END\nCLUSTER_IP_RESOURC
EMANAGER=CLUSTER_IP_RESOURCEMANAGER\nCLUSTER_IP_NAMENODE=CLUSTER_IP_NAMENODE\nCLUST
ER_IP_WORKERS=CLUSTER_IP_WORKERS\nCLUSTER_HOSTNAMES_WORKERS=CLUSTER_HOSTNAMES_WORKE
RS/,"CLUSTER_IP_INIT=#{$ip_init}\nCLUSTER_IP_END=#{$ip_end}\nCLUSTER_IP_RESOURCEMAN
AGER=#{ip_resource_manager}\nCLUSTER_IP_NAMENODE=#{ip_namenode}\nCLUSTER_IP_WORKERS
=#{ips_worker.join(",")}\nCLUSTER_HOSTNAMES_WORKERS=#{hostnames_worker.join(",")}")

changeOnFile("#{path}/cluster/scripts/namenode.sh",
/CLUSTER_IP_INIT=CLUSTER_IP_INIT\nCLUSTER_IP_END=CLUSTER_IP_END\nCLUSTER_IP_RESOURC
EMANAGER=CLUSTER_IP_RESOURCEMANAGER\nCLUSTER_IP_NAMENODE=CLUSTER_IP_NAMENODE\nCLUST
ER_IP_WORKERS=CLUSTER_IP_WORKERS\nCLUSTER_HOSTNAMES_WORKERS=CLUSTER_HOSTNAMES_WORKE
RS/,"CLUSTER_IP_INIT=#{$ip_init}\nCLUSTER_IP_END=#{$ip_end}\nCLUSTER_IP_RESOURCEMAN
AGER=#{ip_resource_manager}\nCLUSTER_IP_NAMENODE=#{ip_namenode}\nCLUSTER_IP_WORKERS
```

```
=#{ips_worker.join(",")}\nCLUSTER_HOSTNAMES_WORKERS=#{hostnames_worker.join(",")}")

changeOnFile("#{path}/cluster/scripts/worker.sh",
/CLUSTER_IP_INIT=CLUSTER_IP_INIT\nCLUSTER_IP_END=CLUSTER_IP_END\nCLUSTER_IP_RESOURC
EMANAGER=CLUSTER_IP_RESOURCEMANAGER\nCLUSTER_IP_NAMENODE=CLUSTER_IP_NAMENODE\nCLUST
ER_IP_WORKERS=CLUSTER_IP_WORKERS\nCLUSTER_HOSTNAMES_WORKERS=CLUSTER_HOSTNAMES_WORKE
RS/,"CLUSTER_IP_INIT=#{$ip_init}\nCLUSTER_IP_END=#{$ip_end}\nCLUSTER_IP_RESOURCEMAN
AGER=#{ip_resource_manager}\nCLUSTER_IP_NAMENODE=#{ip_namenode}\nCLUSTER_IP_WORKERS
=#{ips_worker.join(",")}\nCLUSTER_HOSTNAMES_WORKERS=#{hostnames_worker.join(",")}")

#=================================================
#Create Key/Value array to update hostname in workers
#=================================================

arr_str=""
for i in 0..hostnames_worker.length-1
    # Access the element.
    arr_str = arr_str + "[\"#{ips_worker[i]}\"]=#{hostnames_worker[i]} "
end

changeOnFile("#{path}/cluster/scripts/worker.sh", /LIST/, "#{arr_str}")

puts "Core cluster network configuration injection success ..."

#================================
#Create onegate and init.sh images
#================================

%x[oneimage create --type CONTEXT --path "#{path}/cluster/scripts/onegate" --name
onegate -d files]
id_image_onegate = getIDbyName("onegate")

%x[oneimage            create            --type            CONTEXT            --path
"#{path}/cluster/scripts/resourcemanager.sh" --name resourcemanager.sh -d files]
id_image_init_resourcemanager = getIDbyName("resourcemanager.sh")

%x[oneimage create --type CONTEXT --path "#{path}/cluster/scripts/namenode.sh" --
name namenode.sh -d files]
sleep (5)
id_image_init_namenode = getIDbyName("namenode.sh")

%x[oneimage create --type CONTEXT --path "#{path}/cluster/scripts/worker.sh" --name
worker.sh -d files]
id_image_init_worker = getIDbyName("worker.sh")

#============================
#Set permission to the images
#============================

sleep(2)

%x[oneimage chmod onegate 644]
%x[oneimage chmod resourcemanager.sh 644]
%x[oneimage chmod namenode.sh 644]
%x[oneimage chmod worker.sh 644]

puts "Onegate and inits scripts created, Resource Manager Context image ID
#{id_image_init_resourcemanager},       Namenode       Context       image       ID
#{id_image_init_namenode}, Worker Context image ID #{id_image_init_worker} and
Onegate Image ID #{id_image_onegate}"

#==========================
#Update IDs templates for vms
#==========================

#---------------
#OS Base Server
#---------------
changeOnFile("#{path}/cluster/templates/vms/panacea_hadoop_resourcemanager",
/IMAGE_ID=IMAGE_ID/, "IMAGE_ID=#{id_image_resourcemanager}")

changeOnFile("#{path}/cluster/templates/vms/panacea_hadoop_namenode",
/IMAGE_ID=IMAGE_ID/, "IMAGE_ID=#{id_image_namenode}")

changeOnFile("#{path}/cluster/templates/vms/panacea_hadoop_worker",
/IMAGE_ID=IMAGE_ID/, "IMAGE_ID=#{id_image_worker}")
```

```
#-------
#Network
#--------
changeOnFile("#{path}/cluster/templates/vms/panacea_hadoop_resourcemanager",
/NETWORK_ID = NETWORK_ID/, "NETWORK_ID=#{id_network}")

changeOnFile("#{path}/cluster/templates/vms/panacea_hadoop_namenode", /NETWORK_ID =
NETWORK_ID/, "NETWORK_ID=#{id_network}")

changeOnFile("#{path}/cluster/templates/vms/panacea_hadoop_worker",  /NETWORK_ID  =
NETWORK_ID/, "NETWORK_ID=#{id_network}")

#---------------
# CD-room Images
#----------------
changeOnFile("#{path}/cluster/templates/vms/panacea_hadoop_resourcemanager",
/IMAGE_ID=IMAGE_INIT_ID/, "IMAGE_ID=#{id_image_init_resourcemanager}")

changeOnFile("#{path}/cluster/templates/vms/panacea_hadoop_namenode",
/IMAGE_ID=IMAGE_INIT_ID/, "IMAGE_ID=#{id_image_init_namenode}")

changeOnFile("#{path}/cluster/templates/vms/panacea_hadoop_worker",
/IMAGE_ID=IMAGE_INIT_ID/, "IMAGE_ID=#{id_image_init_worker}")

changeOnFile("#{path}/cluster/templates/vms/panacea_hadoop_resourcemanager",
/IMAGE_ID=IMAGE_ONEGATE_ID/, "IMAGE_ID=#{id_image_onegate}")

changeOnFile("#{path}/cluster/templates/vms/panacea_hadoop_namenode",
/IMAGE_ID=IMAGE_ONEGATE_ID/, "IMAGE_ID=#{id_image_onegate}")

changeOnFile("#{path}/cluster/templates/vms/panacea_hadoop_worker",
/IMAGE_ID=IMAGE_ONEGATE_ID/, "IMAGE_ID=#{id_image_onegate}")

puts "Updated OS ID's, Network ID, CD-Room ID's for VM's templates"

#====================================
#Create the templates for the cluster
#====================================

id_templ_resourcemanager            =            %x[onetemplate             create
"#{path}/cluster/templates/vms/panacea_hadoop_resourcemanager"]
id_templ_resourcemanager = id_templ_resourcemanager.delete('^0-9')

id_templ_namenode               =               %x[onetemplate                create
"#{path}/cluster/templates/vms/panacea_hadoop_namenode"]
id_templ_namenode = id_templ_namenode.delete('^0-9')

id_templ_worker                 =                %x[onetemplate                create
"#{path}/cluster/templates/vms/panacea_hadoop_worker"]
id_templ_worker = id_templ_worker.delete('^0-9')

puts "Templates created, Resource Manager template ID #{id_templ_resourcemanager},
Namenode template ID #{id_templ_namenode}, Worker template ID #{id_templ_worker}"

#====================================
#Update service cluster template file
#====================================

changeOnFile("#{path}/cluster/service/cluster.json",          /WORKER_CARDINALITY/,
"#{num_workers}")
changeOnFile("#{path}/cluster/service/cluster.json",          /WORKER_TEMPLATE_ID/,
"#{id_templ_worker}")
changeOnFile("#{path}/cluster/service/cluster.json",        /NAMENODE_TEMPLATE_ID/,
"#{id_templ_namenode}")
changeOnFile("#{path}/cluster/service/cluster.json", /RESOURCEMANAGER_TEMPLATE_ID/,
"#{id_templ_resourcemanager}")

#================================
#Create the VM's for the cluster
#================================
wait(id_image_worker)
wait(id_image_namenode)
wait(id_image_resourcemanager)

id_service_templ             =             %x[oneflow-template             create
"#{path}/cluster/service/cluster.json"]
id_service_templ = id_service_templ.delete('^0-9')
```

```
sleep(2)
id_service = %x[oneflow-template instantiate #{id_service_templ}]
id_service = id_service.delete('^0-9')
sleep(2)

#============
#Final result
#============
text = %x[oneflow show #{id_service}]
puts text
puts "\n"
puts "Deploy made succesfully, right now you set up a Hadoop Cluster with
#{num_workers} workers ..."
puts "\n"
puts "Wait until all the machines are ready, use the command 'onevm list' to
check..."
```

It is also visible in the bash script above that we make some 'injections' of parameters to the some other bash scripts residing inside of each of the different roles in the cluster, i.e. NameNode, ResourceManager and Worker, this as we have mentioned before are already included in our test images and execute when they are first booted and they are in charge of the different actions need to be performed so that all of the daemons and configurations are correct and running. An example of this code is provided below and it belongs to the NameNode, similar code is found for the ResourceManager and Worker but they change other required parameters which are not included due to space constraint.

```
#*****************************************************
# Mount OpenNebula context variale and enable Onegate
#*+*****************************************************
cp /mnt/onegate /usr/bin
chmod +x /usr/bin/onegate

TOKENTXT=`cat /mnt/token.txt`
echo "export TOKENTXT=$TOKENTXT" >> /home/hadoop/.bashrc

function export_rc_vars
{
        if [ -f $1 ] ; then
                ONE_VARS=`cat $1 | egrep -e '^[a-zA-Z\-\_0-9]*=' | sed 's/=.*$//'`

                . $1

                for v in $ONE_VARS; do
                    echo "export $v=\"${!v}\"" >> /home/hadoop/.bashrc
                done
        fi
}

export_rc_vars /mnt/context.sh

#*****************************************************
# Cluster core network configuration for Hadoop Cluster
#*****************************************************
CLUSTER_IP_INIT=CLUSTER_IP_INIT
CLUSTER_IP_END=CLUSTER_IP_END
CLUSTER_IP_RESOURCEMANAGER=CLUSTER_IP_RESOURCEMANAGER
CLUSTER_IP_NAMENODE=CLUSTER_IP_NAMENODE
CLUSTER_IP_WORKERS=CLUSTER_IP_WORKERS
CLUSTER_HOSTNAMES_WORKERS=CLUSTER_HOSTNAMES_WORKERS

#******************************
# Change IP of NameNode
#******************************
if [ -n "$CLUSTER_IP_NAMENODE" ]; then
     sudo sed -i "s/127.0.1.1/$CLUSTER_IP_NAMENODE/g" /etc/hosts
     sudo service hostname restart; sudo service networking restart
fi

#*************************
# Strict Host Key Checking
#*************************
sudo -H -u hadoop bash -c 'touch /home/hadoop/.ssh/config'
sudo -H -u hadoop bash -c 'chmod 644 /home/hadoop/.ssh/config'
```

```
sudo -H -u hadoop bash -c 'echo "Host *" >> /home/hadoop/.ssh/config'
sudo -H -u hadoop bash -c 'echo "          StrictHostKeyChecking no" >>
/home/hadoop/.ssh/config'

#*******************************************************************************
*******
#    Cluster    configuration    in    /etc/hosts    and    /usr/local/hadoop/hadoop-
2.6.0/etc/hadoop/slaves
#*******************************************************************************
*******
IFS=',' ;

#----- declare an array variable ------
declare -a ips=($CLUSTER_IP_WORKERS)
declare -a hosts=($CLUSTER_HOSTNAMES_WORKERS)

#----- get length of an array ------
num_workers=${#ips[@]}

#------ use for loop read all values and indexes ----
#----- get length of an array ------
num_workers=${#ips[@]}

#------ use for loop read all values and indexes ----
echo -en '\n' >> /etc/hosts
echo '# Cluster network configuration' >> /etc/hosts
echo -en '\n' >> /etc/hosts
echo "$CLUSTER_IP_RESOURCEMANAGER      Hop-ResourceMan" >> /etc/hosts

echo    '#    Cluster    network    configuration'    >>    /usr/local/hadoop/hadoop-
2.6.0/etc/hadoop/slaves
echo -en '\n' >> /usr/local/hadoop/hadoop-2.6.0/etc/hadoop/slaves

for (( i=1; i<${num_workers}+1; i++ ));
do
  echo "${ips[$i-1]}      ${hosts[$i-1]}" >> /etc/hosts
  echo "${hosts[$i-1]}" >> /usr/local/hadoop/hadoop-2.6.0/etc/hadoop/slaves
done

#*****************************
# Format Namenode to use HDFS
#*****************************
su -c 'bash /usr/local/hadoop/hadoop-2.6.0/bin/hdfs namenode -format' - hadoop

#*****************************
# Start Namenode HDFS
#*****************************
su -c 'bash /usr/local/hadoop/hadoop-2.6.0/sbin/start-dfs.sh' - hadoop
```

ANNEX 4

Example of code adapted from Hadoop 1.x to Hadoop 2.x

```
//package /*org.atosworldgrid.*/daaas.pocjob;

import org.apache.hadoop.conf.Configured;
// New one for 2.6
import org.apache.hadoop.conf.Configuration;
// Ends
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxMinAverageDriver extends Configured {


	public static void main(String[] args) throws Exception {
		if (args.length != 2) {
			System.err.printf("Usage: %s [generic options] <input> <output>\n");
		}
		//Deprecated Job job = new Job(getConf(), "Sensor Max, Min and Average diary
values");
		//New one starts here
		Configuration conf = new Configuration();
		Job job = Job.getInstance(conf, "Sensor Max, Min and Average diary values");
		//Ends here
		job.setJarByClass(MaxMinAverageDriver.class);
		FileInputFormat.addInputPath(job, new Path(args[0]));
		FileOutputFormat.setOutputPath(job, new Path(args[1]));
		job.setMapperClass(SensorDataMapper.class);
		job.setReducerClass(MinMaxAverageSensorDataReducer.class);
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(FloatWritable.class);
		System.exit(job.waitForCompletion(true) ? 0 : 1);

	}

}
```