Open Access Library Journal, 2014, 1, 1-10
Published Online June 2014 in OALib. http://www.oalib.com/journal
http://dx.doi.org/10.4236/oalib.1100526

OALib

Assessment, Design and Implementation of a Private Cloud for MapReduce Applications

M. Salgueiro[1], P. González[1], T. F. Pena[2], J. C. Cabaleiro[2]

[1]Department of Electronics and Systems, University of A Coruña, A Coruña, Spain
[2]Centro de Investigacións en Tecnoloxías da Información, CITIUS University of Santiago de Compostela, Santiago de Compostela, Spain
Email: marcos.salgueiro@gmail.com, patricia.gonzalez@udc.es, tf.pena@usc.es, jc.cabaleiro@usc.es

Received 24 April 2014; revised 30 May 2014; accepted 10 June 2014

Abstract

Scientific computation and data intensive analyses are ever more frequent. On the one hand, the MapReduce programming model has gained a lot of attention for its applicability in large parallel data analyses and Big Data applications. On the other hand, Cloud computing seems to be increasingly attractive in solving these computing problems that demand a lot of resources. This paper explores the potential symbiosis between MapReduce and Cloud Computing, in order to create a robust and scalable environment to execute MapReduce workflows regardless of the underlaying infrastructure. The main goal of this work is to provide an easy-to-install interface, so as non-expert scientists can deploy a suitable testbed for their MapReduce experiments on local resources of their institution. Testing cases were performed in order to evaluate the required time for the whole executing process on a real cluster.

Keywords

MapReduce, Hadoop as a Service, OpenStack, Private Clouds, Cloud Computing, Virtualization

Subject Areas: **Big Data Search and Mining, Cloud Computing**

1. Introduction

Scientific Computing enables to perform new kind of experiments that would have been impossible only a decade ago. Nowadays, Big Data science is generating datasets that are increasing exponentially in both complexity and volume, making their analysis a big challenge. Two issues should be addressed: finding an effective method to tackle such challenging problems, and obtaining the necessary resources to solve them.

How to cite this paper: Salgueiro, M., González, P., Pena, T.F. and Cabaleiro, J.C. (2014) Assessment, Design and Implementation of a Private Cloud for MapReduce Applications. *Open Access Library Journal*, **1**: e526.
http://dx.doi.org/10.4236/oalib.1100526

MapReduce [1] may help in addressing the first issue. The MapReduce programming model abstracts the common difficulties linked to distributed processing on large clusters, by offering a simple and efficient way of processing large data sets with a parallel distributed algorithm. Although it has been argued that MapReduce does not suit well for many scientific algorithms, a recent work [2] studied how to adapt different classes of algorithms into the MapReduce model and concluded that the MapReduce programming model can be used successfully even for solving complex scientific computing problems.

As for the second matter, Cloud Computing [3], that agglutinates miscellaneous subsystems forming a unified interface to flexibly deploy and manage virtual clusters, seems to suit well in solving scientist's resource problems. Nowadays, most of the computing power of an institution is spread in different environments. Even though the computational capacity of the institution as a whole may grow, the resources available to each individual user remain very limited. Besides, provided that many machines may remain idle for long periods of time, the distributed computing environment can be under-used and inefficient. Previous projects [4] have already studied the scope of establishing private clouds at the universities. With these clouds, students and researches can efficiently use the already existing resources of university computer networks in solving computationally intensive scientific problems.

The major contribution of this work is *qosh* (*quick-openstacked-hadoop*), a simple and unified interface to manage MapReduce computations, leveraging any existing IaaS (*Infrastructure as a Service*) deployment with a little customization, while providing an automatic one node test installation based on OpenStack [5] and Apache Hadoop [6].

The structure of the paper is as follows. Section 2 briefly reviews related work. Section 3 elaborates on current IaaS and MapReduce framework implementations, focusing on its highlights and drawbacks. Section 4 details qosh architecture and self-installing deployment structure, and goes through a complete execution cycle, identifying key points of the process. Section 5 reviews qosh performance when deployed on a real cluster. Finally, Section 6 collects a reflection on qosh main contributions and future development guidelines.

2. Related Work

Installing an IaaS cloud framework is not an easy task. Some solutions, such as DevStack [7], ease the complexity to install and configure cloud environments. However, they do not provide a usable environment to develop applications on top of an IaaS cloud.

Since MapReduce and Cloud Computing together may prove useful in servicing a potential world of data consumers, it is easy to understand the growing interest in integrate both technologies. Currently, the best known example of a unified approach to said technologies is Amazon Elastic MapReduce (EMR) [8]. Nonetheless, there are other implementations focusing on extending EMR functionality, either by surpassing its constraints—information must be made semi-public and MapReduce workflows need to be executed on Amazon's installation—with Resilin [9], Sahara [10] or Dynamic MapReduce [11], or by reusing its cloud interface to build a MapReduce platform upon like with Cloud MapReduce [12].

None of the previous solutions presents such a simple and integrated approach of installation, configuration and operation as the one presented in this work. All of them involve some extra effort from the user, requiring, in general: knowledge of its inner workings, a previous cloud deployment, the payment for consumption, or the requisite of making public the input data to the MapReduce framework. In summary, the solution presented in this paper may be the natural choice for initially small-scale deployments or as an introductory point to MapReduce and cloud IaaS technologies. From a minimum and automatic deployment, the cloud administrator can add new processing nodes, update the Hadoop virtual machine, alter the mechanics of provisioning or even install another cloud provider without too much effort.

3. Frameworks and Software Selection

This Section discusses about the selection of frameworks and software to be used in the design and implementation of our solution.

In order to abstract virtual cluster creation and destruction, qosh relies on an IaaS cloud framework. Even though there are a good number of these frameworks, they all share a common architecture and cover a similar set of functionality with mixed maturity levels. Structurally, they are comprised of a series of modules connected together by an asynchronous message broker. Internally, they save their processing information in a da-

tabase and exploit their server hardware through the use of a hypervisor. Externally, they expose their capabilities implementing a REST interface to be consumed by a demanding client.

It could be argued that any listing that covers the most widely used IaaS cloud frameworks must include OpenStack [5], CloudStack [13], OpenNebula [14] and Eucalyptus [15]. Precisely, in order to determine which of them could couple qosh best, a reduced installation was carried out before putting them to use. The testing methodology considered diverse subjective magnitudes, such as documentation completeness, installation complexity, modular flexibility, standardization, etc., to give a general view of each one of them. Keeping in mind both the end users and the variability of the field, the target were stable frameworks, that do not change substantially over time, well documented, and with a solid community behind to bring support. In the end, OpenStack came up on top, inasmuch as the latest two releases have immensely improved both its reach in real deployments and its perceived functional maturity.

In the last decade, many encoding frameworks for the MapReduce paradigm have appeared, such as Apache Hadoop, GridGain [16], or Twister [17], among others. Of these, Hadoop is unquestionably the MapReduce framework most widely used today. Its open source nature and its flexibility, both of processing and storage, have reported growing interest from the IT industry. The qosh's development relies on the Hadoop framework.

The goal of this work is to provide an easy-to-install interface, so as non-expert scientists can deploy a suitable testbed for their experiments on local resources of their institution. Thus, the qosh setup defaults to a single node installation in which both infrastructure and execution environment are configured. **Figure 1** precisely depicts the layered configuration. Atop Fedora 17, a setup script downloads and installs OpenStack precompiled packages, and afterwards it downloads, untars and registers a virtual machine image containing an Oracle 1.7 JRE and Apache Hadoop 1.0.4 installation. Likewise, it automatically creates the right user and tenant so that qosh may be put to use straightaway.

At the right end of **Figure 1**, it appears an *Interface* module lying on top of Fedora and being connected to both OpenStack and Hadoop. Its main purpose is to deploy virtual Hadoop clusters, to manage its component virtual machines (VMs) lifecycles and to orchestrate MapReduce workflows executions.

4. Architecture and Implementation

The qosh installation script will automatically configure a highly-performing testing environment that could be easily scaled-out as demand grows. **Figure 2** represents the layered setup decomposition in a single node after the installation procedure had finished.

The OpenStack modules deployed are those fundamentally required by a minimum standalone setup [5]:

Keystone manages authorization, authentication and quota by user and tenant.

Nova handles VMs lifecycles and networking configuration, routing and data flow utilizing Kernel Virtual

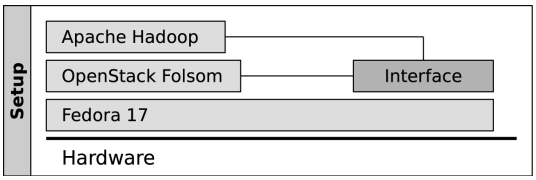

Figure 1. High level design diagram.

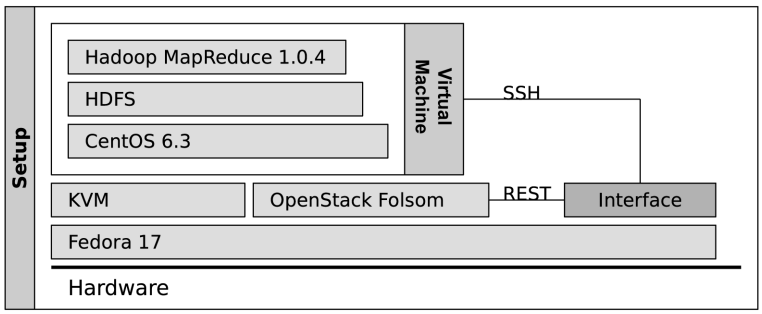

Figure 2. Layered initial deployment.

Machine (KVM) as hypervisor.

Glance holds the browsable catalog of installed VM images on the local file system.

This implies that no fault tolerance measures are defined—as expected from a single node and local file system arrangement—cloud-wise, but it certainly allows for other standard safety protocols to be implemented—e.g. some RAID level with replication or UPS solutions.

4.1. Interface

Figure 3 represents the user interface modular composition. There are three essential modules within: Compute, Fabric, and Django.

4.1.1. Compute

Compute is the REST access client that bridges the OpenStack cloud with the web interface, effectively decoupling qosh from the infrastructure provider. It basically encapsulates a series of methods by which an authorized user is allowed to manually define VM deployment behavior.

Current implementation manages virtual clusters defined with OpenStack running on a single real cluster, i.e. no hybrid clouds are supported. However, *Compute* may be effortlessly adapted to handle VMs running on other IaaS deployments or to manage hybrid clouds, with no interaction whatsoever with another module, as far as qosh API semantics are preserved.

4.1.2. Fabric

Fabric is a Python library used to simplify the management of our virtual cluster by establishing SSH tunnels with the VMs, letting qosh shape Hadoop configuration, putting processing data into HDFS (Hadoop Distributed File System) and recovering results to user space; everything as SSH traffic.

To establish SSH connections our *Fabric* module is fed a *Keystone*-generated keypair. This keypair is created on each virtual cluster deployment and shared by all VMs in the same cluster. Its private part is injected into VMs once they have finished booting, and its public part is kept on the local file system. It is automatically removed—both from OpenStack and file system—when Hadoop execution completes.

4.1.3. Django

Django glues together both modules, renders HTML to be displayed to the user, and organizes result and metadata storage.

Django can be plugged different back-ends, from session objects managers to static file storage, to deal with varying needs and to accommodate future demands. The qosh plugin configuration includes: MySQL, used as meta-information repository; the server file system, to save and retrieve MapReduce I/O data; and *OpenStack-Backend* to delegate into *Keystone* user access and quota.

Putting it all together, a user would define MapReduce computations though a Django-backed web interface. Django would pass configuration parameters on to *Fabric* for creating and feeding input data to a virtual Hadoop cluster. And lastly, real infrastructure would be provisioned by an IaaS cloud driven through *Compute* module.

4.2. Deployment

The qosh installation script will take care of a single node deployment in an automatic fashion, so no previous knowledge of OpenStack or Hadoop would be required to exploit qosh elastic MapReduce prowess in this case; though the virtual cluster elasticity would be heavily constrained. To overcome this limitation, qosh has been architected to abstract the infrastructure underneath, allowing for any IaaS framework to be deployed at any size (some parts of the *Compute* module would require rewriting, nonetheless, if OpenStack were not used).

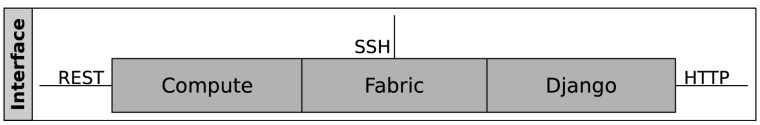

Figure 3. Interface composition.

An installation may be grown from a starting single node setup just by laying out a real IaaS cloud cluster of any size. In fact, any public cloud Amazon Elastic Compute Cloud-compatible (EC2-compatible) could be used to expose infrastructure that qosh would utilize to spawn virtual clusters of any size.

4.3. Apache Hadoop Virtual Machine

The Apache Hadoop installation has been manually configured from scratch inside a virtual machine. It has been conceived to have a minimum footprint while maintaining a server-grade stability. In order to fulfill these requirements CentOS was chosen and an EC2-compatible VM was built up with it [18].

An EC2-compatible VM differs from a *regular* VM in a few peculiarities:

- Container format: It is really subject to framework requirements but the most commonly preferred formats are raw and qcow2 for cloud images, while traditional VMs depend exclusively on the virtualization platform support.
- User access: It is controlled by injecting a private part of an SSH keypair into booting VMs, so that only users with the public counterpart are allowed to log in. However, that private part is not *pushed* into the VM file system by the cloud framework itself, it is *pulled* instead to a web location, concealed from other VMs, so it is the VMs duty to fetch and safeguard that keypair.
- VHDD resizing: Which is the ability to change, on demand, the HDD size of the VMs, can only be accomplished if *direct kernel boot* was being used. Enabling a VM image to boot a kernel directly implies extracting both initram and kernel images from the VM file system and uploading them to the particular cloud framework deployed.

Bearing those singularities in mind, an Apache Hadoop and Oracle JRE installations, a limit in what kind of SSH connections can be established—only those authenticated by keypair—and a final compression, together, yielded qosh, which has potential to be executed on any cloud distribution (considering it being EC2-compatible).

4.4. Execution Flow

Before any MapReduce processing take place, a user should log in into qosh web interface and navigate to the *Define Job* page. **Figure 4** contains a visual representation of a complete execution cycle starting up from that point.

1) When *Define Job* is completely rendered, a form to configure a new MapReduce job and its supporting virtual cluster is presented to the user.

2) In case the form be correctly filled, all of the input data and configuration parameters would be uploaded server-side.

3) Once the upload have finished, a new process is spawned to manage the remaining procedure; meanwhile, the user is sent back to the *Home* page.

4) To guarantee a fair level of privacy, an SSH keypair is created anew on each MapReduce execution. Along with it, a set of virtual machines, or instances, is started.

5) As the amount of time required to bring up networking on each instance varies depending on virtual and real cluster size, a mechanism to check their networking status had to be devised. In order to reduce the complexity and coupling introduced by making the instances fire a *networking-ready* signal, an improvement is introduced. Instead of pushing a *ready* event from the VM to the cloud, the process supervising their creation is kept looping trying to establish an SSH connection to the instances, up to a certain number of attempts.

6) Once every instance can be reached through SSH, a virtual Hadoop cluster is configured following the guidelines contained in *Fabric* script.

7) Through *Fabric* mediation, Hadoop daemons are started on every instance, input data and workflow implementation are pushed onto HDFS and the MapReduce application is started.

8) When the job is finished, the results will be fetched from the virtual cluster to the local file system, where they will be permanently stored.

9) Lastly, the instance set is destroyed and the keypair removed from both OpenStack and the local file system.

5. Performance Evaluation

To assess a measure on qosh performance, a custom deployment was carried out on two nodes. In one of them,

the automatic installation procedure was executed first and the resulting configuration tweaked later to communicate with the other node. In the other, the bare minimum required to allow for VM execution was manually installed (OpenStack *Compute* and required libraries).

Both machines share the same physical configuration which is comprised of an octo-core Intel Xeon CPU layout, 8 GB RAM, 200 GB SATA 3 HDD and dual Gigabit Ethernet connectivity.

5.1. Testing Methodology

In order to give a general picture of qosh performance, three different testing cases were set up to measure the time required by the whole executing process. Timing marks have been *hard coded* to cut relevant parts off and to reduce measurement errors. To that account—to bound the experiment variance, testing cases were executed and measured ten times per deployment configuration; displayed results reflect a simple average of said measures.

Five components were gauged:

- *Deploying time*, stands for the time interval elapsed from the instant that the virtual cluster is starting to be spawned up to when all of the instances can be reached. It should be noted this component adds up to a second of error per instance, due precisely to the one second delay between retries as explained in Section 4.4.
- *Configuring time*, alludes to Hadoop configuration time requirement. It includes transferring and decompressing input data, pushing them to HDFS and laying Hadoop configuration out in the virtual cluster.
- *MapReducing time*, covers exclusively the amount required to complete the MapReduce job.
- *Cleaning time*, spans just the temporal lapse that is needed to remove the keypair and to shutdown every instance in the cluster.
- *Total time*, is the time it took the whole process to complete. Note that it does not equal partial times summation.

The MapReduce application used is the well-known WordCount, which counts the number of words in an actual document set. Apache Hadoop was configured to allocate to its underlaying JVM up to the maximum RAM available to the instance.

The first test case centered on evaluating qosh timings when the virtual cluster was scaled out. Thus, a fixed 62.5 MB plain text document list was fed to a growing virtual cluster ranging from one 1 GB RAM/VCPU instance to eight (4 instances on each node, as OpenStack was laid over a dual-node cluster).

For the second test case, the number of VMs spawned remained constant in different executions but its capabilities—RAM and VCPU count—were sequentially doubled from 1 GB RAM/VCPU to reach 4 GB RAM/VCPU, to account for qosh tendency on vertical scaling. The 62.5 MB of plain text was kept untouched as on the first case.

Finally, a third test case presents the resulting measurements obtained on the same virtual cluster configuration, when the input size was sequentially doubled from 62.5 to 250 MB.

5.2. Results Analysis

Figure 5 presents the evolution of different timings as instance count increases from one to eight. It can be seen how deploying, configuring and clearing measurements rise slightly with instance count, while processing time drops, all as expected.

In **Figure 6** it is shown how vertical scaling affects qosh performance. Deploying and cleaning times stay constant (two instances on every execution), whereas processing and configuring times are reduced as instances are upgraded.

Looking at both charts it can be seen that, for our particular dual-node deployment, vertical scaling uses infrastructure resources more efficiently: two 4 GB RAM/VCPU instances are *roughly* equal to eight 1 GB RAM/VCPU, but total executing time is reduced by almost 28%.

Finally, **Figure 7** shows timing evolution as input size heightens. In this case, the time of MapReduce increases with the size of the problem, and also the configuration time due to the copy of data files in the HDFS.

6. Conclusions and Future Work

In this paper, a simple and unified interface for deploying OpenStack Framework and Hadoop MapReduce

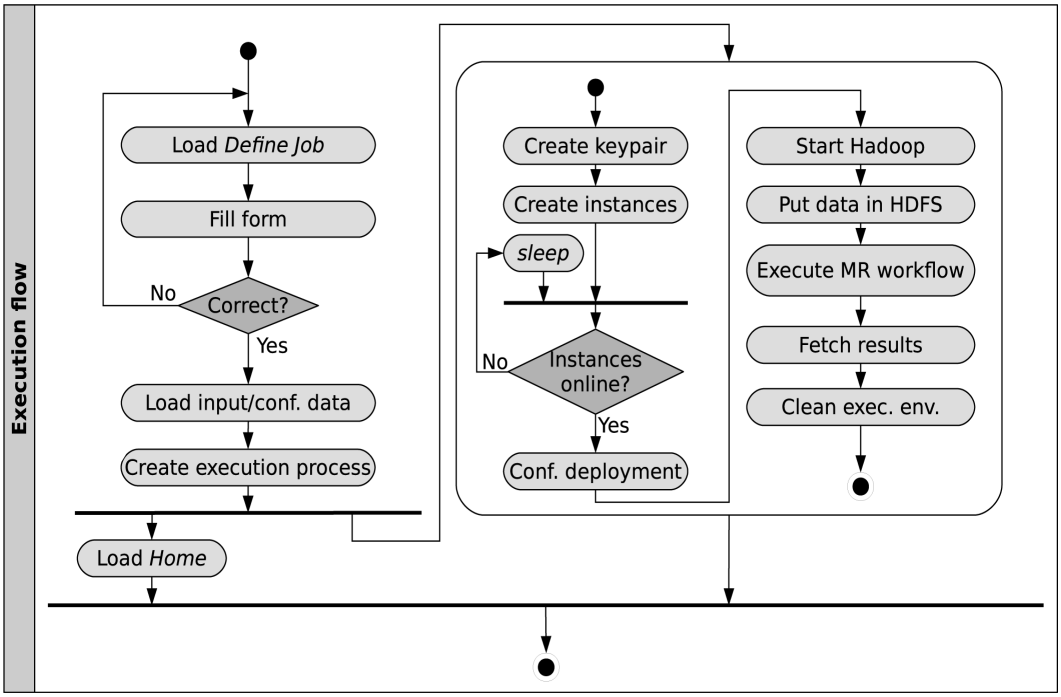

Figure 4. Global execution flow.

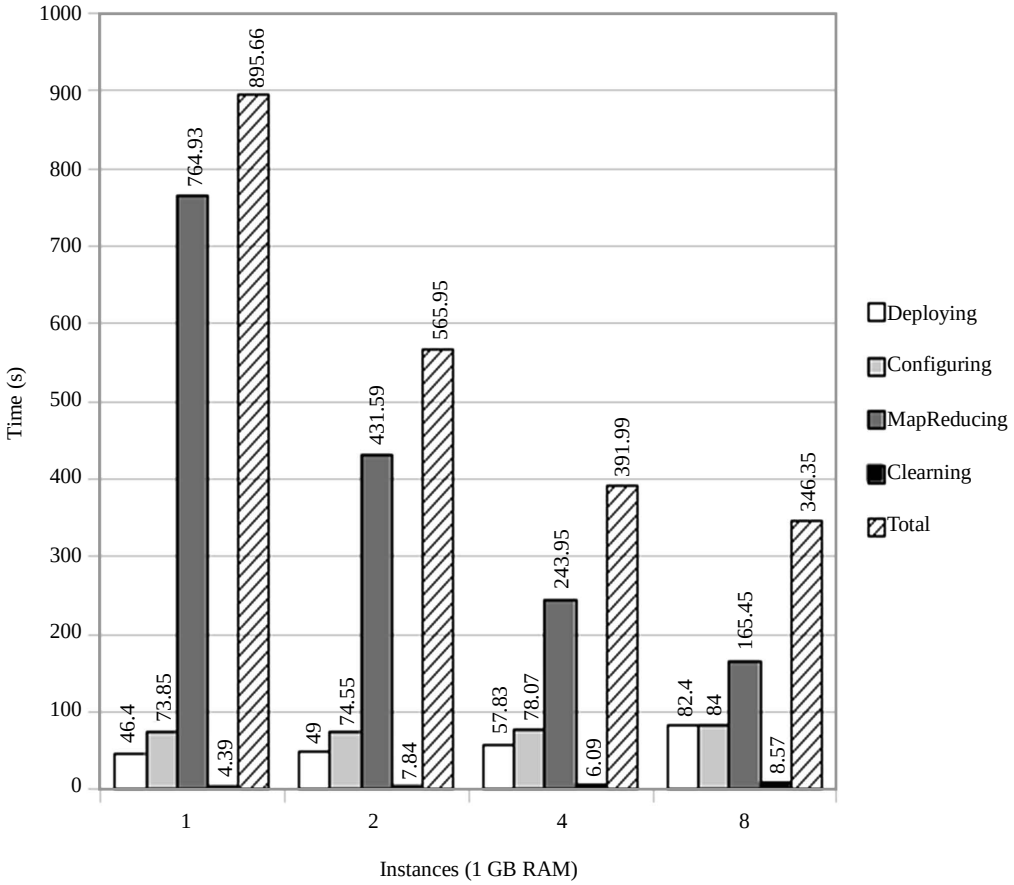

Figure 5. Tendency on scaling out.

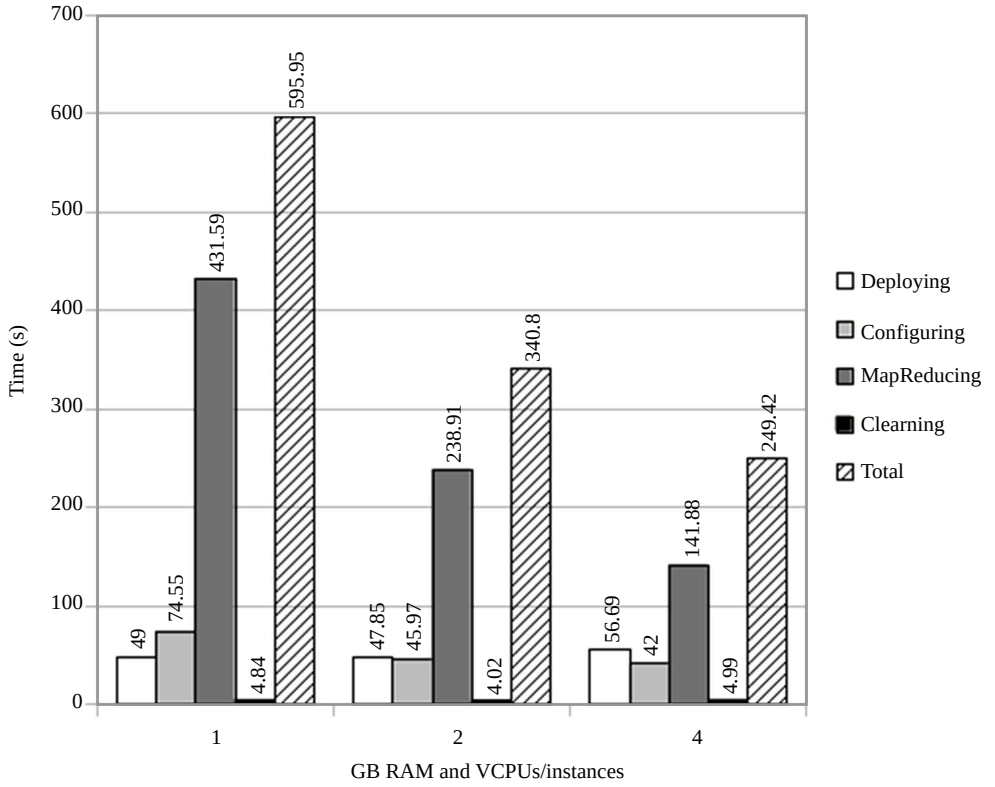

Figure 6. Tendency on scaling in.

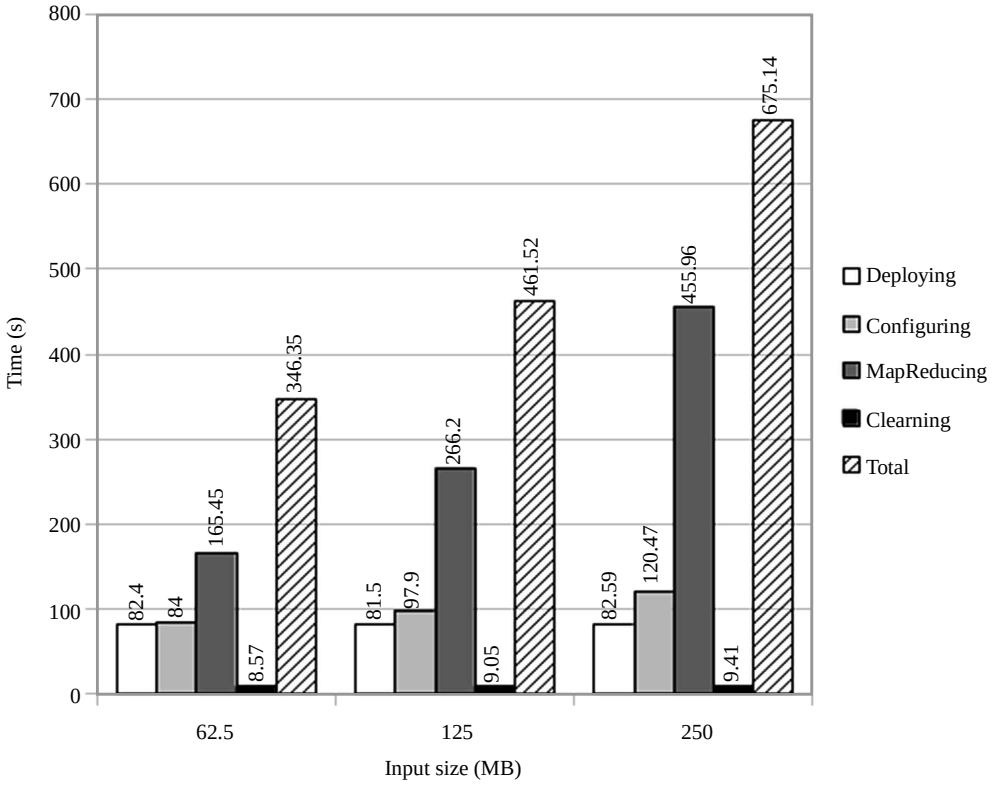

Figure 7. Tendency on input size escalation.

together building a private cloud is presented. The solution main contributions can be summarized in the following points.

- Simplicity of installation and exploitation. The qosh installation script is straightforward and its web interface provides the means to define custom virtual Hadoop deployments, to launch MapReduce jobs and to download results easily and intuitively.
- Vertical integration of every module, from the web interface to file storage and infrastructure provisioning. The qosh installation script configures automatically a complete execution environment with no user intervention required.
- High performance on initial setup, since qosh has been conceived from the beginning to execute MapReduce workflows on virtual clusters, while maintaining a minimum learning curve.
- Reusability of the main components. Adhering to *Amazon Web Services* standards when building the VM allows deploying it over any EC2-compatible cloud IaaS. Additionally, its worth considering the ability to make the VM run with no cloud architecture underneath, by setting up an installation on top of a real cluster.
- Adaptability to handle infrastructure provided by other cloud frameworks, and thus, opening the possibility to spread virtual clusters on hybrid clouds just by rewriting the *Compute* module.
- Source code and documentation both from qosh and its required libraries and applications, are publicly available online [19].

Future Work

The main limitation of qosh is a certain coupling between modules. It would be interesting to abstract a REST client interface in order to code different *delegates*, which could be used to adapt messages to any REST API language, clearing hybrid cloud implementations up.

Following the same decoupling dynamics, some use cases like view processing in *Django* could be detached from the web interface as action objects, and be executed in an action processor. This action processor could be shaped, for instance, a processes pool consuming these action objects queuing in memory easing off horizontal scaling and load balancing.

References

[1] Dean, J. and Ghemawat, S. (2008) MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, **51**, 107-133. http://vgc.poly.edu/~juliana/courses/cs6093/Readings/dean-cacm2008.pdf

[2] Ekanayake, J., Pallickara, S. and Fox, G. (2008) MapReduce for Data Intensive Scientific Analyses. *IEEE Fourth International Conference on eScience*, Indianapolis, 7-12 December 2008, 277-284.

[3] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I. and Zaharia, M. (2009) Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

[4] Srirama, S.N., Jakovits, P. and Vainikko, E. (2012) Adapting Scientific Computing Problems to Clouds Using MapReduce. *Future Generation Computer Systems*, **28**, 184-192. http://dx.doi.org/10.1016/j.future.2011.05.025

[5] OpenStack. http://www.openstack.org

[6] White, T. (2009) Hadoop: The Definitive Guide. O'Reilly Media.

[7] DevStack. http://devstack.org

[8] Amazon Web Services: Elastic MapReduce. http://aws.amazon.com/elasticmapreduce

[9] Riteau, P., Iordache, A. and Morin, C. (2011) Resilin: Elastic MapReduce for private and community Clouds. Research Report RR-7767, INRIA.

[10] OpenStack: Project Sahara. https://wiki.openstack.org/wiki/Sahara

[11] Loughran, S., Alcaraz Calero, J.M., Farrell, A., Kirschnick, J. and Guijarro, J. (2012) Dynamic Cloud Deployment of a MapReduce Architecture. *IEEE Internet Computing*, **16**, 40-50. http://dx.doi.org/10.1109/MIC.2011.163

[12] Liu, H. and Orban, D. (2011) Cloud MapReduce: A MapReduce Implementation on Top of a Cloud Operating system. 11*th IEEE/ACM International Symposium on Cluster*, *Cloud and Grid Computing*, Washington DC, 23-26 May 2011, 464-474.

[13] Apache Cloudstack. http://cloudstack.apache.org

[14] Moreno-Vozmediano, R., Montero, R.S. and Llorente, I.M. (2012) IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. *IEEE Computer*, **45**, 65-72. http://dx.doi.org/10.1109/MC.2012.76

[15] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Youseff, L. and Zagorodnov, D. (2009) The Eucalyptus Open-Source Cloud-Computing System. 9*th IEEE International Symposium on Cluster Computing and the Grid*, Shanghai, 18-21 May 2009, 124-131.

[16] GridGain Systems. GridGain 3.0—High Performance Cloud Computing Whitepaper. Technical Report, 2011.

[17] Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-H., Qiu, J. and Fox, G. (2010) Twister: A Runtime for Iterative MapReduce. *Proceedings of the* 19*th ACM International Symposium on High Performance Distributed Computing*, Chicago, 21-25 June 2010, 810-818.

[18] Apache Hadoop 1.0.4 Based on CentOS 6.3 VM. https://drive.google.com/file/d/0B2lmVzXW-C5UcmZIYk80dTZJb0k/edit?usp=sharing

[19] Qosh Main Page. https://code.google.com/p/quick-openstacked-hadoop