# Dynamic Deployment of a MapReduce Architecture in the Cloud

Steve Loughran, Jose M. Alcaraz Calero, Andrew Farrell, Johannes Kirschnick, Julio Guijarro

**Abstract**—Recently cloud-based MapReduce services have appeared to process large data sets in the Cloud, significantly reducing users' infrastructure requirements. Almost all these services are Cloud vendor-specific and thus internally designed within their own cloud infrastructure. This leads to two important limitations. Cloud vendors do not provide any clue about how they manage the MapReduce architecture internally hampering its evaluation and also users are not able to either build their own private cloud infrastructure based offering or to use different public cloud infrastructures for this purpose. Thus, this paper describes an architecture which enables the dynamic deployment of a MapReduce architecture in virtual infrastructures provided by either public or private cloud providers. This architecture has been implemented and validated as a proof of concept and released to the community.

**Index Terms**—Automated Deployment, MapReduce, Cloud Computing, Infrastructure as a Service, Configuration Management

## 1 Introduction

THE processing of large data sets is becoming more and more important in research and business environments. Researchers are demanding tools to quickly process large amounts of data and businesses are demanding new solutions for data warehousing and business intelligent, using large data sets. For these problems, the MapReduce [1] programming model has demonstrated its value in many scenarios. Probably, the most successful example is the indexing system that produces the data structures used for the Google [1] web search service.

The main challenge associated with processing large data sets is the required infrastructure to carry out the processing. Usually, the infrastructure demands large up-front investments to cope with the highest anticipated workload and this investment needs to be justified by the actual usage of the platform. However, the demand for the processing platform can be very fluctuating, creating times of over and under utilization. Thus, Cloud computing can significantly reduce the infrastructure capex, providing new business models in which *infrastructure providers* provide on-demand virtualized infrastructures in a pay-as-you go model. In this new model, infrastructure can be dynamically adapted to the data processing requirements of the *infrastructure consumer* offering an elastic infrastructure for data processing.

Recently, some cloud providers started to provide services for large data processing as part of their offerings, based on the MapReduce programming model. A prominent example is the *Amazon Elastic MapReduce* service. While *MapReduce as a service* is an important advantage in the field of large data processing, it exposes several limitations. Firstly, cloud providers offer such services in a ready-to-use fashion (i.e. as Platform-as-a-Service) and they do not provide any details about how this service is implemented and how it works internally, e.g. how the deployment, configuration and execution of MapReduce Jobs is carried out. This fact hampers the evaluation of the efficiency of such services and can be directly related to the hard task of developing tools for deploying efficiently large distributed systems like MapReduce. Secondly, clients do not have control over the MapReduce software stack and its configuration, which may lead to optimization, performance and compatibility problems. Thirdly, these services are always vendor-specific tailored to their own infrastructure, preventing clients to use multiple cloud providers, their own private cloud infrastructure or even for offering their own Elastic Cloud-based MapReduce service. Note that the usage of private clouds can be especially relevant when sensible information is processed, which is an open issue nowadays for public clouds.

The main purpose of this paper is to describe an architecture which enables the dynamic deployment of a MapReduce architecture in virtual infrastructures provided by either public or private cloud providers. This architecture overcomes the previously described limitations since it enables the usage of different combinations of public and private cloud providers, exposes transparently how the MapReduce service is managed and enables the customization of such service. The deployment of the MapReduce service is performed using SmartFrog [2], a configuration management software which hides the complexity involved in the MapReduce service provisioning whilst retaining the full control over individual aspects of the service. Hadoop is the MapReduce implementation used for validating the architecture provided.

This paper is structured as follows. Section 2 introduces the MapReduce programming model for processing large data sets. After that, section 3 provides an introduction to a suitable architecture for cloud infrastructure provider, followed in section 4 by an overview of our architecture to enable the dynamic provisioning of a MapReduce service in such infrastructure provider. Implementation details are

- *J.M. Alcaraz-Calero is with Department of Communications and Information Engineering, University of Murcia, Computer Science Faculty, Murcia, Spain, 30100.*
  *E-mail: jmalcaraz@um.es*
- *S. Loughran, J.M. Alcaraz-Calero, A. Farrell, J. Kirschnick and J. Guijarro are affiliated with the Cloud and Security Lab, Hewlett Packard Laboratories, BS34 8QZ Bristol, United Kingdom.*
  *E-mails: steve.loughran@hp.com, jose.alcaraz-calero@hp.com, andrew.farrell@hp.com, johannes.kirschnick@hp.com, julio.guijarro@hp.com*

described in section 5; Moreover, statistical results are provided in section 6. Related works of cloud services for data management are discussed in section 7. Finally, we draw conclusions in section 8.

## 2 MapReduce Programming Model

The MapReduce programming model [1] provides a framework for process large data sets in parallel. Figure 1 provides a high level overview of the execution steps performed by the framework.

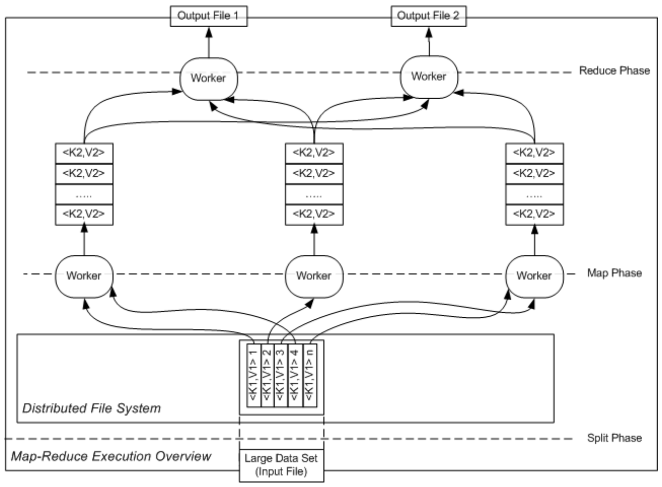

Figure 1. MapReduce Execution Overview

The initial large data set is split into processing chunks according to a pre-defined *split function*, which defines the elemental processing units, such as a database row, a column, a line of a file. This function must generate <key,value> pairs as in <position of the line in a file, content of the line>.

Each result of the split is assigned to a *worker*. At this stage, the workers are designated to perform the *map* phase. Here the specified *map* function, provided by the developer, produces from the input stream intermediate <key,value> pairs. These tuples are automatically grouped and sorted by their *key* and forwarded to the *reduce* phase. There a worker applies the *reduce* function on all values associated with a particular key. As a result, each worker produces a partial output of the data which is usually stored in an output file. An optionally specified a *merge* function can be applied to join the outputs from two or more reducers.

Note that developers only have to specify the *split*, *map*, *reduce* and *merge* functions avoiding them from having to deal with aspects related to parallel programming. The framework takes care of scheduling, monitoring and check-pointing of individual jobs.

The architecture is conceptually a master-slave one, with many slave nodes processing data (workers) orchestrated by a master node in charge of assigning, controlling, and synchronizing all of the slave nodes jobs. We refer the reader to [1] for more information on the MapReduce programming model.

## 3 Infrastructure as a Service

An *Infrastructure as a Service* (IaaS) cloud provider offers computation and storage resources to third parties. IaaS provider act as resource brokers, providing access to their infrastructure and services in a pay-as-you-go model, leveraging virtualization to enable secure resource sharing. Figure 2 shows a conceptual design of an IaaS with a set of typically offered core services.

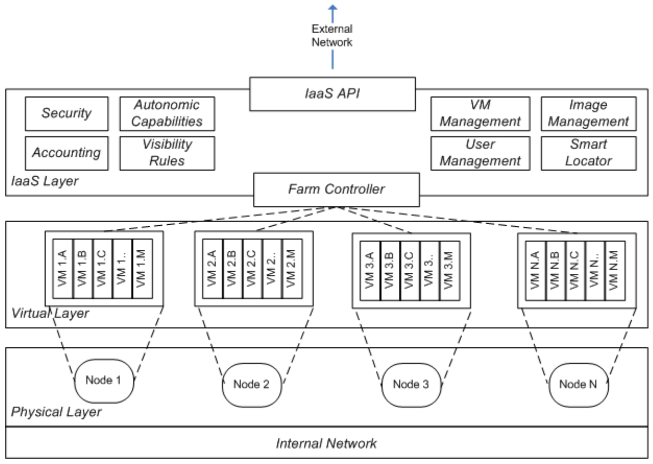

Figure 2. Infrastructure as a Service

The physical layer encompasses all computational resources found in one or more data centres. The virtualization layer enables the secure isolated sharing of these resources and the IaaS layer is in charge of managing these virtual resources efficiently. The core services used by an IaaS are typically: i) user management; ii) controlled access to the infrastructure resources; iii) accounting for usage of resources; And, iv) ability to create virtual infrastructures on-demand;

It is worth noting the *IaaS API* in Figure 2. This API gives third parties access to infrastructure services via the Internet, and is used by our cloud service to automatically create the infrastructure needed for the MapReduce architecture.

## 4 Manage Large Data Sets using MapReduce in the Cloud

This section provides a detailed description of our proposed architecture for carrying out the dynamic deployment of a MapReduce service in elastic virtual infrastructures. Figure 3 shows the main components: An *Elastic MapReduce Service* offering data processing as a service for third party applications through an API, and, above that, a web-based GUI providing direct access for users.

The web-based component is a user-friendly front-end for the services provided by the *Elastic MapReduce Service*. The service itself is composed of several layers, each of them described in the following subsections. The main components for monitoring and deployment are flexible enough to be deployed in any type of machine, physical or virtual, potentially located either at the client side or in a public cloud. The deployment in a public cloud leads to a similar offer as provided by Amazon Elastic MapReduce but using a white-box approach enabling users to know what is really happening underneath. Hosting it client-side enables the deployment into multiple private and public cloud providers. Private and public cloud providers each have different connectivity requirements, with the public clouds to be more restrictive due to the low bandwidth, firewalled communications, data security, etc.

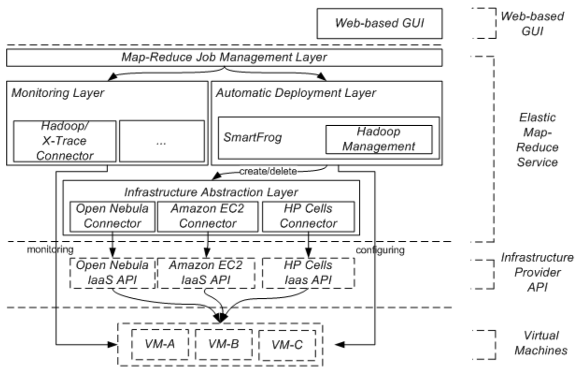

Figure 3. Architecture of the Cloud Service for Data Management using MapReduce

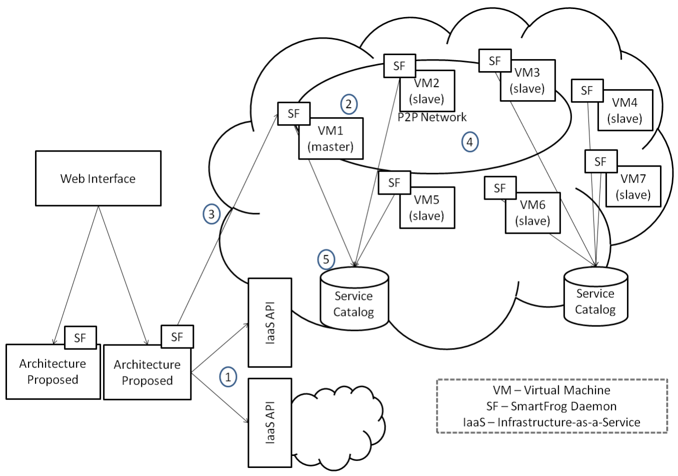

Figure 4. Overview of the Automatic Deployment Process

In the following we assume a scenario in which the framework is deployed at client-side and the intensive data processing is done in a public cloud. However, note that this framework enables the exclusive usage of private clouds for processing sensible information.

## 4.1 MapReduce Job Management Layer

The *MapReduce Job Management Layer*, depicted in Figure 3, exposes the only access point for users to the proposed Elastic MapReduce service via an HTTP REST interface. This interface lets users create, configure and execute new jobs and offers abilities to set parameters related to the configuration of each job.

Firstly, the virtual infrastructure parameters for the data processing are captured, i.e. number of master and slave nodes, the cloud provider to use and boot volume to utilized. Secondly, input files and output folder locations are captured. The input files have to be uploaded separately using the API, if they are not already present in the deployed infrastructure. Thirdly, the job configuration parameters, i.e. selected $split$, $map$, $reduce$ and $merge$ functions are captured and finally, the cloud provider credentials, i.e. user/pass, token, etc. All this user provided configuration parameters are further referred to as configuration parameters. This is the only user interaction needed, the rest of the process is done by the proposed framework automatically.

After capturing the input data the job is executed. This triggers the deployment of the virtual infrastructure using the *Automatic Deployment Layer*. To track and monitor running MapReduce jobs services provided by the *Monitoring Layer* can be used.

Finally, this layer also provides the capability to define complex jobs as dataflows in which several jobs are chained together to achieve complex data processing flows. Such workflows can then be submitted to the *MapReduce Job Management Layer* for execution.

## 4.2 Automatic Deployment Layer

This layer performs the deployment of the virtual infrastructure, installation and configuration of the MapReduce implementation and the execution of MapReduce jobs. The user triggers this process by starting a new job. It uses the job configuration parameters captured by the Job Management Layer. Figure 4 shows an overview of the steps involved in the automated deployment.

Firstly, The required number of VMs (master and slave nodes) are created and started within the selected cloud provider. Here the *Infrastructure Provider Abstraction Layer* is used to abstract the specifics of individual providers by presenting a common high level interface. (See #1 in Figure 4)

The VMs are driven by the boot volume attached to them. The majority of cloud providers use a preloaded boot volume which may contain a preconfigured MapReduce implementation for slave and master nodes, i.e. a static provisioning of the MapReduce service. This approach does not fit very well with public cloud environments due to: i) It requires manual maintenance, as each image update requires downloading, mounting, updating and uploading the image (several GBs) across the Internet. It also hampers with keeping the MapReduce implementation up-to-date. ii) All non-default, job specific configuration parameters have to be manually selected by the user. iii) There is no automated management of the infrastructure and services. iv) There exists no run-time model with the current status of the infrastructure and services.

Our architecture provides a different approach: sharing a boot volume across all VMs. This boot volume consists of a clean OS installation with a Configuration Management Tool (CMT) installed as daemon, which is automatically launched as part of the OS booting process. (See #2 in Figure 4). This CMT daemon is able to carry out the dynamic provisioning of services at run-time, receiving software installation and configuration instructions and performing the execution on the local OS. This approach is used to dynamically creating the MapReduce architecture, thus overcoming the aforementioned shortcomings.

Puppet [3], CHEF [4] and CFEngine3 [5] are client-server CMT solutions originally designed for distributed environments. Although they do a good job at configuring software artifacts, they do not cope well with cloud environments in which virtual infrastructure needs to be created as part of the provisioning of services. SmartFrog (SF) [2] and SLIM [6] have been designed for cloud environments, using a peer-to-peer and a client-server architecture respectively. In our solution we use SmartFrog for several reasons: i) SF is a pure peer-to-peer architecture with no central point of failure.

It enables fault tolerance deployment, critical in intensive data processing in virtual environments, where resources are out of the control of the user and could spontaneously disappear. ii) It facilitates different communication protocols between SF daemons suitable for both private and public cloud environments. iii) It enables dynamic reconfiguration capabilities to change infrastructure and services at run-time stage. iv) It furthermore keeps a model of the current deployment status which can be used to drive auto-scaling based on observed metrics. v) It enables to use late binding configuration information to configure services, information which only becomes available after a process step has been reached, especially needed in cloud environments where users usually do not have control over resource names, e.g. IP address. Here we use it to inform the slave nodes of the IP Address of the Master node.

Once all the VMs are booted and the SF daemons are running, the MapReduce implementation has to be installed and configured. We have used Hadoop, a well-known MapReduce Java implementation. Installation is driven by an configuration file automatically generated, created by the *Automatic Deployment Layer* using the configuration parameters for the given job. In this scenario one of the VMs is randomly chosen to be the master and the others become slaves. The purpose of the master VM is: i) to receive and process the configuration file (See #3 in Figure 4) and ii) to act as the master node of the Hadoop architecture.

The configuration file contains all necessary information to install and configure the whole Hadoop framework and to execute the MapReduce jobs. Figure 5 depicts a simplified version of a generated configuration file using the *SmartFrog* syntax.

```
masterNode extends HadoopMasterNode{ // Install "hadoop-master","hadoop-nodename", "hadoop-dataname" packages
 sfProcessHost "10.0.0.1"; //VM IP
 slaveNodeList LAZY [slaveNode1, slaveNode2]; //List of slave nodes
 running true;
}

slaveNode1 extends HadoopSlaveNode{ // Install "hadoop-slave" package
 sfProccesHost "10.0.0.101"; //VM IP
 masterNodeList LAZY [masterNode]; //List of master nodes
 running true;
}

slaveNode2 extends HadoopSlaveNode { // Install "hadoop-slave" package
 sfProccesHost "10.0.0.102"; //VM IP
 masterNodeList LAZY [masterNode]; //List of master nodes
 running true;
}

job extends MapReduceJob{ // Job specifications
 masterNode LAZY masterNode;
 dataInput     "file://localhost/home/input.dat";
 dataOutput    "file://localhost/home/output/";
 source        "file://localhost/home/wordcount.jar"
 MapClass      "com.hp.cloud.hadoop.Mapper";
 ReduceClass   "com.hp.cloud.hadoop.Reducer";
 SplitterClass "com.hp.cloud.hadoop.Splitter";
}
// Deployment dependencies for the Map-Reduce infrastructure
dependsOn(slaveNode1, masterNode::running == true);
dependsOn(slaveNode2, masterNode::running == true);
dependsOn(job, masterNode::running == true &&
               slaveNode1::running == true &&
               slaveNode2::running == true);

sfConfig extends slaveNode1, slaveNode2, masterNode, job; //Config entry point
```

Figure 5. Simplified examples of SmartFrog configuration file using Hadoop MapReduce implementation

The example file shown in figure 5 defines a master node and two slave nodes. It also contains the dependency information as state dependencies for each component. In this example, the master node has to be *run* before the deployment of the slave nodes starts. Note as well the usage of late binding variables (using the reserved word *LAZY*).

The *Automated Deployment Layer* submits this configuration file to the SF daemon in the master VM, which fragment this file and propagates these fragments across the other SF daemons in the peer-to-peer network (See #4 in Figure 4). For the master node, the associated fragment is processed: installing and starting the Hadoop master node (Nodename and JobTraker). All the slave nodes proceed in a similar manner: upon receiving a file fragment and processing it, SmartFrog installs the Hadoop slave components (Datanode and TaskTracker).

SmartFrog uses application repositories (See #4 in Figure 4) for retrieving the necessary application files. These repositories store the Hadoop packages to be copied and executed in the VMs and the configuration templates filled with the configuration parameters and copied into the VM. In case the reader is more interested about how to deal with the automatically installation and configuration of applications in the cloud, Kirschnick et al [7] provides a comprehensible description. Note that these repositories can be hosted within the cloud provider itself, minimizing the network delay to further reduce the time required for provisioning the VMs

Once Hadoop is running, the input data is uploaded to the internal distributed file system used by Hadoop (HDFS). After that, the MapReduce job is submitted to the Hadoop master node, triggering the execution of the job. Finally, once the MapReduce job has finished, the output files of the job are extracted from Hadoop and copied into the file system used by the service, making them accessible to the user by means of the REST interface.

### 4.3 Monitoring Layer

The *Monitoring Layer* is in charge of tracking the MapReduce jobs and to present this information to users. It monitors the VMs and the running MapReduce components. Monitoring information can either be periodically pulled from the components or pushed to it. To do this, the master VM must run a monitoring software suitable for the specific Hadoop implementation, which exposes job statistics. Different plug-ins can be used to provide additional information depending on the used software stack and cloud provider. For example, the system for monitoring the Infrastructure provider can yields on OS metrics whereas the X-Trace [8] monitoring software can provide specific Hadoop metrics.

### 4.4 Infrastructure Provider Abstraction Layer

The *Infrastructure Provider Abstraction Layer*, depicted in Figure 3, is intended for homogenizing the *IasS API* provided by the different cloud infrastructure providers, presenting a high level view across different public and private infrastructure provider. Additional providers can be added by supplying new *Infrastructure Provider Connector* implementations. These connectors map the *IaaS API* offered by the provider with a homogeneous interface used by the upper layers.

### 4.5 Security Issues

We use existing encryption, authentication, and access control mechanisms to enable secure data processing. Although security in private clouds is less restrictive, the usage of public cloud providers requires maximizing the security in the data exchange between our architecture and the hosting VMs. Then, the *Infrastructure Provider Connector* has to deal with this, using secure transport protocols such as SSH, SFTP and HTTPS when possible. Regarding the boot image, it might have a firewall and anti-virus software installed and the OS might be up-to-date.

To securely process data in outsourced data centers is an open issue nowadays. It is not the intention of this paper to provide a solution for this issue. However, we try to minimize the time in which data is stored in the cloud. Thus, data is never persisted in the boot disk. Input data for the MapReduce job is copied dynamically to the running VMs when necessary. Moreover, once the process has finished, the final results are copied to the local storage (assumed at the client-side) and the VM is cleaned (including MapReduce cache). We take this approach to maximize the privacy of the data against unauthorized accesses (by both external users and the infrastructure provider).

Moreover, we do not share the local file system used in our Elastic MapReduce service between different tenants. The file system is isolated by physically separating it per each tenant or providing an authorization system for controlling the access to the files stored in the local file system, depending on the scenario.

## 5 Implementation

The proposed architecture has been implemented as a proof of concept and released publicly under LGPL[1]. It is composed of two different components: A RESTful web service offering the functionalities of the *MapReduce Job Management* layer and a front-end web-based application. The web application can be be installed in both *Servlet* and *Portlet* containers.

The elastic MapReduce service itself has been implemented as a war artifact which may be deployed into an existing web application server. The boot volume are composed of a clean installation of Ubuntu 9.04 and SmartFrog v3.17. Additional components that SmartFrog relies on have been added to the image. For example, *apt-get* to install necessary Linux packages required by Hadoop. Furthermore the SmartFrog Hadoop component has been pre-installed, which provides the necessary classes used to map configuration parameters to the configuration file format expected by Hadoop and to control individual services (e.g. start, stop. etc.). To test our architecture, we developed a number of different cloud connectors: HP private cloud *HP Cells*, *VMWare*, *Mock* and *Open Nebula* and *Amazon EC2* for public clouds.

## 6 Statistics

This section provides some performance statistics about executing Hadoop in the Cloud. The jobs consists of sorting 4 GB of randomly generated records. This 4GB is spread across 10 files of 400 MB, each generated using the Hadoop $randomWriter$ sample application. Both applications are available in the latest $Hadoop$ release. For comparison, the MapReduce job and its data is fixed while only the number of workers is varied. Different virtual infrastructures have been created ranging from 1 to 50 workers. For statistical relevance, each individual experiment has been repeated 50 times and the values presented are averages over all test runs. This experiment uses *HP Cells* as cloud provider, our internal infrastructure provider, i.e. a exclusive private cloud used only for this experiment (no extra workload). This small cloud is composed of 6 physical blades with an Intel Core 2 Quad and 6GB RAM, 500GB HDD, connected via gigabit lan. Each blade is configure to manage up to 8 VMs.

1. The Elastic Hadoop Service is available at http://smartfrog.svn.sourceforge.net/viewvc/smartfrog/trunk/core/extras/hadoop-cluster/

The intention of this experiment is: i) to measure the relationship between the time for creating the virtual infrastructure and for booting the OS (infrastructure creation time), the time for provisioning and starting the infrastructure with the MapReduce implementations (provisioning time) and the time for executing the MapReduce job, including data uploading and downloading (MapReduce execution time). ii) to validate the architecture for the automatic deployment of Hadoop in the Cloud, since all these tests have been executed using a mere batch script which interacts with the REST interface, configures the jobs, submits them into the framework and finally gathers the time information from the monitoring layer.

Although, this sequence of batch jobs can be executed using the workflow capabilities, we have decided to execute each job in isolation, forcing a re-deploy of the whole infrastructure every time; however, this could be optimized in production by exposing an *do-not-undeploy* parameter to the user which if enabled retains the deployed resources so that they can be reused by the same user for additional jobs, avoiding or minimizing the creation and provision times on the next executions.

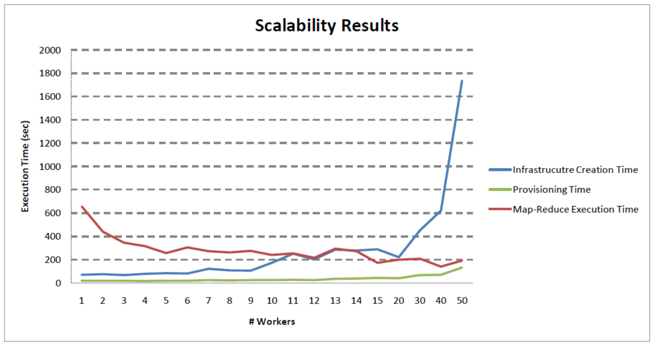

Figure 6. Scalability Results

Figure 6 shows a clear decrease in the time spent executing the MapReduce job whilst increasing the number of available workers. This is an excepted result because the MapReduce framework is built to scale with the available resources. The minimal fluctuation in the linear trend may be due to different placement of the vms in the physical infrastructure creating the needs for additional network hops or not if vms where deployed on the same machine and the sometimes unpredictable behavior of virtualization technologies in shared environments. The provisioning time is also constants and sometime even achieves super linearity due to the SmartFrog capability for doing parallel installation and configuration of applications. Note that on average the overhead time with respect to the infrastructure creation time is around 10% for carrying out the provisioning of services, which keeps it in the boundary of acceptable times.

Both Hadoop execution and provisioning times validates the scalability of the proposed architecture since they are the times directly related to the architecture proposed.

Regarding the infrastructure creation time it is almost constant up to 10 workers. After that, the time grows linearly (note that the X scale steps are not constant) when bigger virtual infrastructures are created. This can be attributed to the small size of our underlying cluster and therefore to the limited number of resources available for the workers. Note that 30 workers means that each physical machine has to support 5 VMs at same time with the associated computational, access disk and memory consumption overhead.

These results show that the virtual infrastructure creation time is very significant with respect to the Hadoop execution time even in a private cloud. This fact emphasizes the importance of giving the final user the ability to decide what cloud provider or private cloud to use if performance is a concern. Finally, note that real workloads will not necessarily work with randomly generated data and may use sensitive data, in these cases the proposed architecture has successfully demonstrated the usage of a private cloud for processing such data, as data it will not cross the security boundaries. The reason for which we have not used a public cloud for this experiment is because we wanted to control that no external data fluctuations due to unpredictable overheads related to the physical machines would yield different deployment times.

## 7 Related Works

Recently, Grossman and Gu [9] explained the design and implementation of a high performance cloud specifically designed to archive, analyze and mine large distributed data sets. In this paper, the authors remark on the advantages of using cloud infrastructure for processing large data sets.

Keahey et Al. [10] describe a cloud platform offering targeted at scientific and educational projects intended for making easy experiments on EC2-style cloud providers. The authors remark that the use of Hadoop on their platform dominated during the whole projects life time and authors associate this fact to the growing interest of the science community and the advantages achieved by combining MapReduce with on-demand cloud infrastructure providers. In fact, Amazon Elastic MapReduce [2] is a well-known service already offering this service. However, this is a black box ready-to-use service which uses Amazon EC2 as cloud provider and does not provide any description about its architecture or enables users to customize the Hadoop software stack. Moreover, they do not provide tools for enabling users to become their own Elastic MapReduce providers, especially for processing sensitive data.

An interesting research very related to our proposal is provided by Liu and Orban [11]. They describe an implementation of the MapReduce programming model on top of the Amazon EC2. This contribution is focused on some specific aspects: how to handle failure detection/recovery and conflict resolution of the MapReduce nodes, controlling latency, tracking jobs, statistics, *etcetera*. However, it does not explain in detail how the MapReduce architecture is deployed, configured and executed. This fact hampers the reproduction and validation of this proposal by other researchers.

2. Amazon Elastic MapReduce is available at http://aws.amazon.com/es/elasticmapreduce/

Mesos [12] is a platform for sharing commodity clusters between multiple diverse cluster computing frameworks, such as Hadoop and MPI. Sharing improves cluster utilization and avoids per-framework data replication. This is a complementary approach to our architecture presented here and could be deployed together with Hadoop enabling data locality by reading and computing data stored on the machine which holds the data and other advantages provided by the framework.

Moreover, other proposals such as Hadoop on Demand (HOD) [3] and the Cloudera Hadoop distribution [4] offer client-side alternatives for devploying Hadoop on Demand. However, none of these solutions cover the on-demand infrastructure creation where Hadoop will be installed as part of the deployment process of the service.

## 8 Conclusions

The MapReduce programming model has shown immense interest for processing large and unstructured data sets. This proposal explains an architecture for automatically deploying Hadoop systems on-demand in the Cloud. This architecture has been validated by means of a prototype implementation. It enables users to interact with the MapReduce programming model while hiding the complexity of deploying, configuring and running the MapReduce software components in the public or private cloud provider involved. This architecture enables users to become their own MapReduce cloud providers since it has been designed as an independent platform-as-a-service layer suitable for different cloud providers. It also enables users to get the full control over the complete data processing since it exposes the SmartFrog configuration system to the final users by means of graphical interfaces to provide a total control, and also enables users to process confidential data safely since allows for the usage of your own cloud infrastructrue.

Regarding future work, it is expected to improve the cloud service with algorithms to automatically schedule tasks using the most suitable cloud provider to maximize the performance while minimizing the price. Moreover, another expected step is to provide auto-scaling capabilities to the architecture according to some business policies as well as dealing with fault tolerance capabilities. We except to adapt the current architecture to the Hadoop NextGen architecture recently announced [5]. Finally, we would also like to explore workflow languages for expressing advanced complex data processing jobs and how these languages can be adapted into the Cloud environment.

## Acknowledgement

Thanks the Funcion Seneca for sponsoring Jose M. Alcaraz Calero under the post-doctoral grant 15714/PD/10.

3. Hadoop on Demand is available at http://hadoop.apache.org/common/docs/r0.17.2/hod.html

4. Claudera Hadoop is available at http://www.cloudera.com/hadoop/

5. The Next Generation of Apache Hadoop MapReduce. http://developer.yahoo.com/blogs/hadoop/posts/2011/02/mapreduce-nextgen

## References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings at 6th Symposium on Operating System Design and Implementation*, 2004.

[2] P. Goldsack, J. Guijarro, S. Loughran, A. Coles, A. Farrell, A. Lain, P. Murray, and P. Toft, "The smartfrog configuration management framework," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 1, pp. 16–25, 2009.

[3] J. Turnbull, *Pulling Strings with Puppet*. FristPress, 2007.

[4] A. Jacob, "Infrastructure in the cloud era," in *Proceedings at International O'Reilly Conference Velocity*, 2009.

[5] M. Burgess, "Knowledge management and promises," *LNCS Scalability of Networks and Services*, vol. 5637, pp. 95–107, 2009.

[6] J. Kirschnick, J. M. Alcaraz-Calero, L. Wilcock, and N. Edwards, "Johannes kirschnick and jose m. alcaraz calero and lawrence wilcock and nigel edwards," *IEEE Communication Magazine*, vol. 48, p. 12, 2010.

[7] J. Kirschnick, J. M. Alcaraz-Calero, P. Goldsack, A. Farrell, J. Guijarro, S. Loughrana, N. Edwards, and L. Wilcock, "Towards a p2p framework for deploying services in the cloud," *Software: Practice and Experience*, vol. (early access on-line), 2011.

[8] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica, "X-trace: A pervasive network tracing framework," in *Proceeding at 4th USENIX Symposium on Networked Systems Design & Implementation*, 2007.

[9] R. Grossman and Y. Gu, "Data mining using high performance data clouds: experimental studies using sector and sphere," in *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008, pp. 920–927.

[10] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, "Science clouds: Early experiences in cloud computing for scientific applications," in *International Conference on Claoud Computing and Its Applications*, 2008.

[11] H. Liu and D. Orban, "Cloud mapreduce: a mapreduce implementation on top of a cloud operating system," Accenture Technology Labs, Tech. Rep., 2009.

[12] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *8th USENIX Symposium on Networked Systems Design and Implementations*, 2011.