West University of Timisoara

Elastic MapReduce in cloud federations

Student:
Ancuta Iordache

Scientific advisors:
Ph.D. Stud. Pierre Riteau
Prof. Dr. Dana Petcu

Abstract

Amazon Elastic MapReduce is a web service for processing large datasets efficiently. It enables the users to write MapReduce applications without having to deal with networking, hardware, instance provisioning, Hadoop configuration and health monitoring. In this way, they are allowed to focus more on the problem to solve. An important limitation of this service is the usage of computing resources provided only by Amazon datacenters at a certain cost.

The goal of this thesis was the creation of a service that provides the same functionalities as Amazon Elastic MapReduce using resources from multiple clouds. In the same time it offers more flexibility as users can choose between different types of virtual machines, operating systems or hadoop versions.

The paper presents the technologies leveraged by our system, followed by a description of the implementation, evaluation results, comparisons with Amazon EMR and improvements that can be done to it.

Contents

Chapter 1

Introduction

1.1 Cloud Computing overview

As a new model derived from grid computing, distributed computing and parallel computing, cloud computing represents the next step in the internet evolution. Nowadays, when searching information on internet or accessing the email, we are using the processing power that relies in a distant location. In fact, even the most basic computer applications require an internet connection to do simple tasks.

The cloud is providing an extension of the user's machine as long as there exists an internet connection, the users have no worries about the processing power or storage requiered by the application, these being provided by data centers. Two of the most significant advantages of this model are the ease-of-use and the cost-effectivenes. The access to the cloud takes place using web browsing capable devices: laptops, desktops, smartphones,PDAs, pad computers etc.

The term of utility computing was mentioned for the first time in 1960 by John McCarthy as "computation may someday be organized as a public utility" thus underlying the concept of cloud computing. The 'cloud' term was used later in 1990 when describing the large Asynchronous Transfer Mode (ATM) networks. Nowadays, because of the lack of a standard definition, there is a fair amount of confusion regarding cloud computing. One of the definitions that covers all the aspects of cloud computing is the one provided by NIST:

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

The main characteristics of the cloud can be extracted from this definition :

- self-service provisioning and automatic deprovisioning when the resources requirements exceed the provider ones;
- broad network access - accessible from anywhere, from any standardized platform (i.e. desktop computers, mobile devices, etc.)
- resource pooling - the computing resources in the cloud are shared thus numerous clients may be using the same set of resources at the same time.
- elasticity - refers to the users ability to add or remove infrastructures resources based on their needs;
- measured service - the cloud provider is measuring the amount of service provided and reacting accordingly (both in terms of billing the client, and updating hardware and software as appropriate)

As the client requires more capacity,he can just buy more storage and computing nodes from another service provider, such that only the consumption is payed, which is cheaper than having to buy and configure all the needed equipment. While the on-demand provisioning capabilities of cloud services eliminates many time delays, there is still a delay as the needs and requirements are determined before capability is automatically provisioned.

Cloud Computing is a support of "everything as a service"(XaaS). The cloud provides means such that everything, from computing power to computing infrastructure, applications, business processes to personal collaboration, can be delivered over the Internet (either as separate components or a complete platform) based on user demand.

The emergence of cloud computing has made a tremendous impact on the Information Technology (IT) industry over the past few years, where large companies such as Google, Amazon and Microsoft strive to provide more powerful, reliable and cost-efficient cloud platforms, and business enterprises seek to reshape their business models to gain benefit from this new paradigm.

1.2 Related technologies

Cloud computing model evolved due to the development of the following technologies, with which it shares some aspects:

- grid computing - distributed computing paradigm that coordinates networked resources to achieve a common computational objective.The use of distributed resources is the common characteristic but cloud computing is leveraging virtualization technologies at multiple levels (hardware and application platform) to realize resource sharing and dynamic resource provisioning;
- utility computing - computing resources like applications, infrastructure and storage are packaged and sold as a service, with users paying only for what they consume, like electricity.
- virtualization - defined as using computer resources to imitate other computer resources or even whole computers. The term is very broad and can include virtualizing everything from memory to software.

1.3 Paper objectives

The goal of this thesis is the implementation of a service that will provide the same functionalities as Amazon Elastic MapReduce but on top of multiple clouds. The service will be in charge of requesting resources from multiple clouds, deploying a MapReduce application and tracking its status. In the remainder of this paper, there will be a description of the cloud computing architecture, the related technologies, the key features and characteristics of it, followed by the description of our application architecture, implementation and evaluation.

Chapter 2

State of the art

2.1 Architecture

The main factors that determined the evolution of cloud computing are the virtualisation technology, the development of universal high-speed bandwidth, and universal software interoperability standards.

2.1.1 Layered model

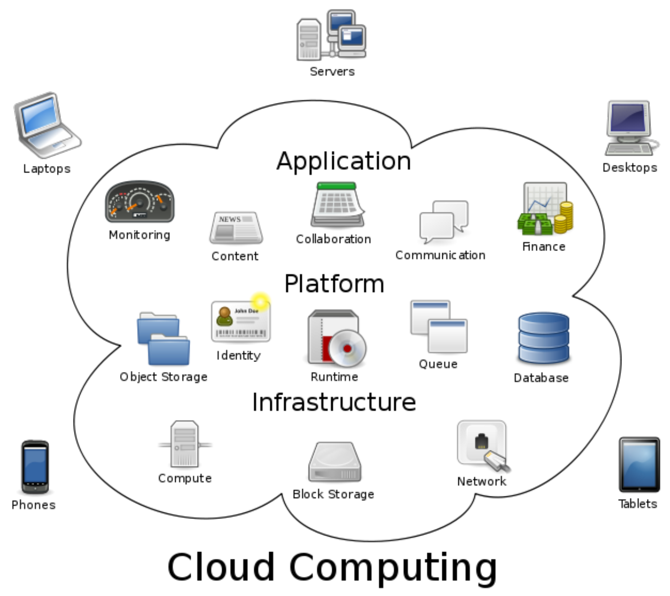
The cloud architecture can be split in 5 layers :

- servers layer - represented by the physical resources of the cloud: physical servers, routers, switches, power and cooling systems.It is typically implemented in data centers which usually contain thousands of servers that are organized in racks and interconnected through switches, routers or other fabrics. Typical issues at hardware layer include hardware configuration, fault-tolerance, traffic management, power and cooling resource management.

- infrastructure layer - virtualization layer, the infrastructure layer creates a pool of storage and computing resources by partitioning the physical resources using virtualization technologies such as Xen, KVM and VMware. The power and performance of commodity x86 hardware continues to increase. Processors are faster than ever, support more memory than ever, and the latest multi - core processors literally enable single systems to perform multiple tasks simultaneously. Virtualization provides an excellent way of getting the most out of existing hardware while reducing many other IT costs.

- platform layer -built on top of the infrastructure layer, the platform layer consists of operating systems and application frameworks. The purpose of the platform layer is to minimize the burden of deploying applications directly into VM containers. For example, Google App Engine operates at the platform layer to provide API support for implementing storage, database and business logic of typical web applications.
- application layer - consists of the actual cloud applications. Different from traditional applications, cloud applications can leverage the automatic-scaling feature to achieve better performance, availability and lower operating cost.
- client layer - computer hardware and/or software used to access the cloud applications. Eg: laptopd, smartphones, browsers.

An architecture overview is presented in the image above:



2.1.2 Business model

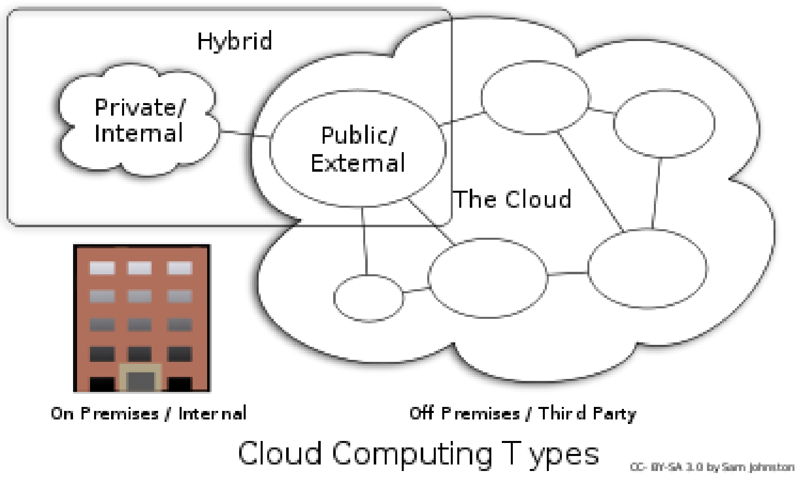
Each layer presented in the previous section can be implemented as a service to the layer above. Practically, clouds offer three categories of services:

- Infrastructure as a Service (IaaS) -refers to on-demand provisioning of infrastructural resources, usually in terms of VMs. The cloud owner who offers IaaS is called an IaaS provider. Examples of IaaS providers include Amazon EC2, GoGrid etc.

- Platform as a Service (PaaS) - level offering virtualised servers on which users can run existing applications or develop new ones without which having to worry about maintaining Operating system,Load Balancing or Computing Capacity;PaaS vendors provide API or development platforms to create and run applications in cloud;Examples of PaaS providers include Google App Engine, Microsoft Windows Azure
- Software as a Service (Saas) - level offering applications over the Internet as a service. Examples of SaaS providers include Salesforce.com, Rackspace, etc.

2.1.3 Deployment models

According to NIST, there are 4 types of clouds : private clouds, public clouds, hibrid clouds and community clouds.



In public clouds, service providers offer their resources as services to the general public, some of the most popular are Amazon and Google apps.

In private clouds,a proprietary network or a data center supplies hosted services to a limited number of people. A private cloud may be built and managed by the organization or by external providers. A private cloud offers the highest degree of control over performance, reliability and security.

A hybrid cloud infrastructure is a combination of public and private cloud models connected by a standard technology that provides data and application portability.

A community cloud infrastructure is shared by several organisations with common concerns. This type of cloud has more users than a private cloud but less than a public one.

2.2 MapReduce overview

2.2.1 Programming Model

MapReduce is a new programming paradigm presented by Google in 2004, for the processing of huge datasets on large commodity clusters. All the computation problems that it applies to, can be expressed by two functions: map and reduce, inspired by the operations from functional languages.

The map function takes as input a key/value pair (which is the basic data unit involved in mapreduce processing) and generates a list of intermediate key/value pairs:

Map(k1,v1) $\longrightarrow$ list(k2,v2)

Then the intermediate pairs are grouped by keys and passed to the reduce function which merges them and returns the new formed list as output:

Reduce(k2, list (v2)) $\longrightarrow$ list(v3)

2.2.2 MapReduce Framework

Since MapReduce was proposed by Google as a programming model for developing distributed data intensive applications in data centers, it has received much attention from the computing industry and academia thus becoming a standard for data analisys. For instance, an open source implementation of mapreduce, Hadoop, has been adopted by a large number of companies as: Yahoo, Facebook, IBM, Amazon, etc. The mapreduce frameworks are processing huge datasets using a large number of computing resources(cluster or grids) and a distributed file system for storing data. The users are submitting a mapreduce job (consisting from a set of tasks) to the scheduling system that will send it for execution on the cluster. Execution consists of three stages:

- map stage when the master node is splitting the input data into blocks of a certain dimension and send them to the slave nodes for processing. The mapper will receive the key/value pair and will generate the intermediate list of key/value pairs.
- shuffle stage when all of the values that are associated with an individual key are sent to the same machine.

- reduce stage when the reducer takes all the values associated with the key and outputs a multiset of key/value pairs with the same key.

The sequential aspect of Mapreduce is observed in the third stage due to the fact that the map stage must be completed before the reduce stage begins, the reducers having all the values for a certain key, they can perform sequential operations on them. The parallelism is involved when reducers that are processing different keys are executed in parallel.

Besides the ease of paralelization, the users don't have to worry about low-level details of parallel programming as : scheduling, fault tolerance ,data distribution and about the shuffle stage, these being handled by the framework.

The disadvantage of this model is that applications must consist only from map and reduce functions, thus users must give up to the programming flexibility for the ease of parallelization .

2.3 Hadoop

The Hadoop project provides an open source framework for the development of highly scalable distributed computing applications, that handles the processing details, leaving developers to focus on application logic. The project is a collection of related subprojects, the one regarding this thesis is the Hadoop Core which provides the basic services for building a cloud computing environment with commodity hardware, and the APIs for developing software that will run on that cloud.

Besides the two fundamental components of Hadoop Core: HDFS (a reliable shared storage) and MapReduce (analysis system) it also provides a set of utilities that support the other Hadoop subprojects. Hadoop Core includes FileSystem, RPC, and serialization libraries.

2.3.1 Hadoop Core

The utilities provided by the Hadoop Core are the following:

- Hadoop commands invoked by the bin/hadoop script. Some of the most important commands provided are user commands as: (distcp for copying files in parralel between filesystems), jar (for running a jar application in the hadoop cluster), job (for getting the status of the submitted jobs), fs(for running a generic filesystem user client) and administration commands:

  - starting/stopping hdfs and mapreduce daemons: namenode, secondarynamenode, datanode, jobtracker, tasktracker
  - for managing HDFS: dfsadmin
  - for managing mapreduce: mradmin

- File System shell for the interaction with HDFS and other file systems supported by Hadoop as Local, HFTP, S3 etc. It is invoked by the command:

  bin/hdfs dfs <args>

  Most of the FS shell commands are similar with the Unix commands, the arguments are URIs with the format:

  scheme://autority/path

  where scheme (for HDFS is hdfs,for the local file system is file) and authority are optional, if not provided it will take the default file system from configuration xml. An examples of folders or files in HDFS is hdfs://namenodehost/parent/child or simply /parent/child (given that hdfs configuration is set to point to hdfs://namenodehost).

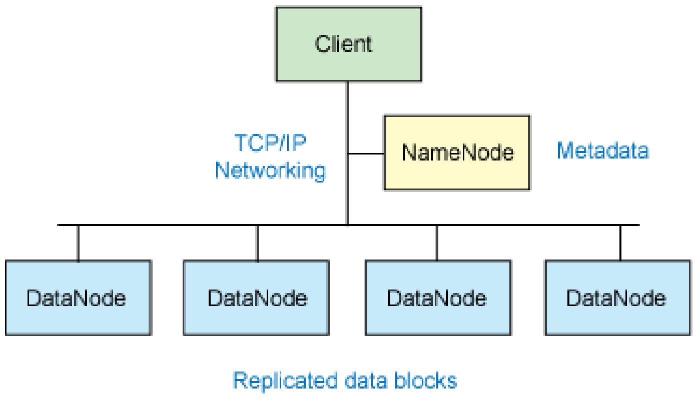
- Service Level Authorization is in charge with clients access permissions to the Hadoop services, being performed before other access control checks such as file-permission checks, access control on job queues etc. The access control lists are defined in the conf/hadoop-policy.xml configuration file.

- Native Hadoop Libraries providing native implementations of certain components for performance reasons and for non-availability of Java implementations.

2.3.2 Hadoop HDFS

HDFS is a distributed file system for storing large files that are hundreds of megabytes, gigabytes, or terabytes in size, designed to run on commodity hardware. The files are broken into block-sized chunks, which are stored as independent units.

One of the most important features of HDFS is data replication, meaning that data is going to be copied to multiple storage nodes in the cluster. As long as there is one replica available, the computation will proceed without notifications about storage failures.



The simplified overview of HDFS architecture in the image above shows the processes in charge with the HDFS system:

- Namenode - it is running on the master node, handling management of the file system metadata and controling the datanodes.

  The management of the filesystem metadata is exerted by maintaining the filesystem tree and the metadata for all the files and directories in the tree, information stored persistently on the local disk in two files: the namespace image and the edit log.

  The datanode controls is exerted by replication, deletion and creation of files on the storage nodes that are running the datanode daemon (meaning that when a file is requested over another file, the namenode will ask the datanodes to replicate it more often for the increasing of availability)

  For decreasing the amount of traffic on the backbone of the network, the replication goes first into the same network switch/rack thus structuring the filesystem in an efficient manner.

  The namenode also knows the datanodes on which all the blocks for a given file are located, however, it does not store block locations persistently, since this information is reconstructed from datanodes when the system starts. Without the namenode, the HDFS cannot be used, all the files on the filesystem would be lost since there would be no way of knowing how to reconstruct the files from the blocks on the datanodes.

- SecondaryNamenode - usually it runs on a separate machine,its main purpose is to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large, keeping a copy of the namespace image for using it in case the namenode is failing.

- Datanodes - representing the workers of the filesystem.

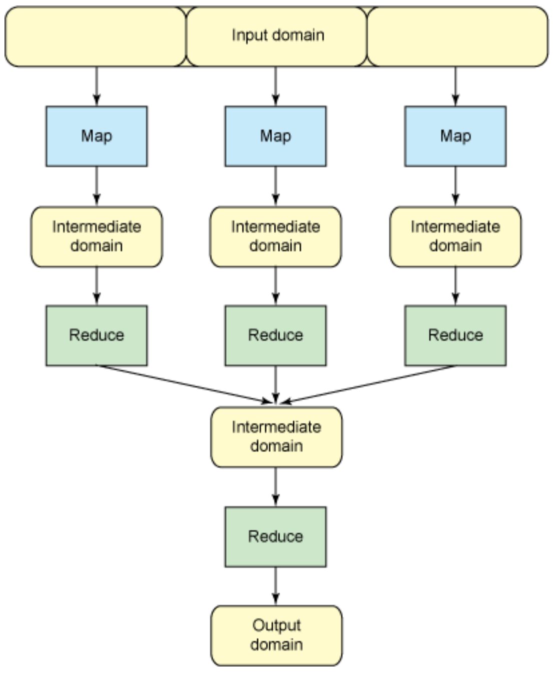
  The Datanodes are responding to read/write commands from HDFS clients and to block creation, deletion and replication commands from Namenode. The Datanodes are sending periodically heartbeat messages to the Namenode, each hearbeat containing the list of blocks stored on the datanode.

  In case of Datanode failure(it won't send the hearbeat to the Namenode), the Namenode is re-replicating the blocks lost on that Datanode.

2.3.3 Hadoop MapReduce

It provides the framework that handles the execution of map reduce tasks across a cluster of machines. When a job is executed, the input data is split into chunks that represent the input of the map tasks, being executed in parallel.Then the output of map tasks is sorted by the framework and forwarded to the reduce tasks for the second phase of execution. The input and the output of the job are usually stored into a filesystem as HDFS. The framework handles the task scheduling, monitoring and re-execution in the event of failures.

A MapReduce execution workflow is presented in the image below:



For starting a map reduce job, the following parameters must be specified (depending on the job, some of them are optional): the jar file containing the map and reduce functions,job input location in the HDFS,job output location in the HDFS, input format, output format, class containing the map function,class containing the reduce function.

The management of the MapReduce job is handled by two processes:

1. JobTracker - runs on the master node, its purposes are the job scheduling and monitoring of tasks on the TaskTracker nodes. It also validates requested work and, if a datanode fails for some reason, reschedules the previous task.

2. TaskTracker - manages the execution of individual map and reduce tasks on a compute node in the cluster.

The only point of failure is the jobtracker, while the failures of tasktracker nodes are handled by the jobtracker.

Chapter 3

Amazon Web Services

Amazon is the most widely known cloud vendor, offering a collection of services accesible over HTTP that provide cloud-based computation, networking, storage and other functionalities enabling the clients to deploy applications on an on-demand provisioning and commodity prices.

3.1 Amazon EC2

Amazon EC2 is an infrastructure service that provides resizable compute capacity in the cloud. It enables cloud clients to start and control virtual servers in a datacenter through the API and tools it provides.

The compute capacity is controlled through a simple web interface, allowing clients to control each virtual server like a physical machine (install and configure software as nedeed). They can create a resizeable pool of servers for handling computing tasks. Users can start a number of virtual servers requiered to perform a task, scaling up or down (adding or removing machines) based on a demand or terminating the machines when the task is completed.

In addition to scalability of resources number, the computing power can be also scaled by using different types of virtual servers (less or more powerful). Only the used resources are going to be billed.

The EC2 service has 3 main components:

- EC2 instances - the virtual machines that are performing the computing task designed to run on physical machines
- virtual machine images - AMIs
- the environment - provides the virtual machines configuration and control

EC2 Instances are virtual machines running on top of Xen virtualization engine meaning that the underlying computer hardware is virtualized and can be adjusted to give performance within defined specifications, regardless of what real physical hardware in Amazon's data centers.

Even if the instances are not running on hardware in the usual sense, they are configured to provide a well-defined amount of computing power.For example, Amazon offers several types of instances providing different level of performance and resourcing, different operating systems. The CPU rating is expressed in ECU (EC2 Compute Unit), the measure unit of processing power defined by Amazon. An instance with rating of 1 ECU offers the same CPU capacity as a physical machine with a 1.0 - 1.2 GHz AMD Opteron processor. The standard instance types characteristics are presented in the table below:

Resource	Small	Large	Extra Large
Platform	32-bit x86	64-bit x86	64-bit x86
CPU rating	1 ECU (1 virtual core)	4 ECUs (2 virtual cores of 2 ECUs each)	8 ECUs (4 virtual cores of 2 ECUs each)
RAM	1.7 GB	7.5 GB	15 GB
Storage	150 GB	840 GB (2X420 GB)	1680 GB (4X420 GB)
I/O Performance	Moderate	High	High
Instance Type Name	m1.small	m1.large	m1.xlarge

The price of these instances are different, corresponding to the computing power offered and the region where the datacenter resides. Prices for EC2 instances from EU (Ireland) region:

	Linux usage	Windows usage
Small (Default)	$0.095 per hour	$0.12 per hour
Large	$0.38 per hour	$0.48 per hour
Extra Large	$0.76 per hour	$0.96 per hour

In case the user does not specify the instance type to launch, the service is going to use the small type, which is the default.

For the creation of an EC2 instance, there must be specified an AMI which is a preconfigured operating system and software. When the instance is launched, it boots from the image's filesystem and runs the configurations and the software stored in the AMI. Then, the user is able to log into the virtual machine as the root and customize it.

EC2 provides access control over instances using SSH protocol, based on RSA keypairs.

3.2 S3

Amazon S3 is a highly reliable distributed storage infrastructure. It offers a very simple data model for storing files:buckets and objects. The bucket is a container holding multiple objects that store data and metadata. Each object has an unique key assigned that is used for retrieving it.

S3 is built with a minimal set of functionalities:

- read, write, delete objects using either a REST-style or SOAP interface. Objects can also be retrieved using the HTTP GET interface or via BitTorrent. An access control list restricts who can access the data in each bucket.

- objects are stored and retrieved using the unique keys

- objects can be public or private, and access can be granted to specific users

S3 provides access control mechanisms, allowing the user to make the data public or private, and grant access to specific users.

Resources from S3 are identified usind URIs with the format http://s3.amazonaws.com/bucket-name/object-name, but it can also be accessed using alternative domain names.

S3s design aims to provide scalability, high availability, and low latency at commodity costs. S3 users are billed monthly for their usage of the service. There are three aspects that are taken into consideration when billing: the storage space consumed, the volume of data transferred to or from S3, and the number of API request operations that have been performed on the account. The prices differ based on the location of S3 bucket.

3.3 Amazon EMR

Amazon Elastic MapReduce is a web service built on top of Amazon EC2 and S3 for processing large data sets. It is enabling users to write mapreduce applications without having to worry about networking, hardware, instance provisioning, Hadoop configuration and health monitoring allowing them to focus more on the problem to solve.

Easy-to-use due to the web console, the Amazon EMR service allows the jobflow to scale up or down based on its need, being also a reliable service by automatically re-executing the failed tasks. Another benefit of Amazon EMR is the cost efectiveness by monitoring the jobflow and turning off resources when it is done.

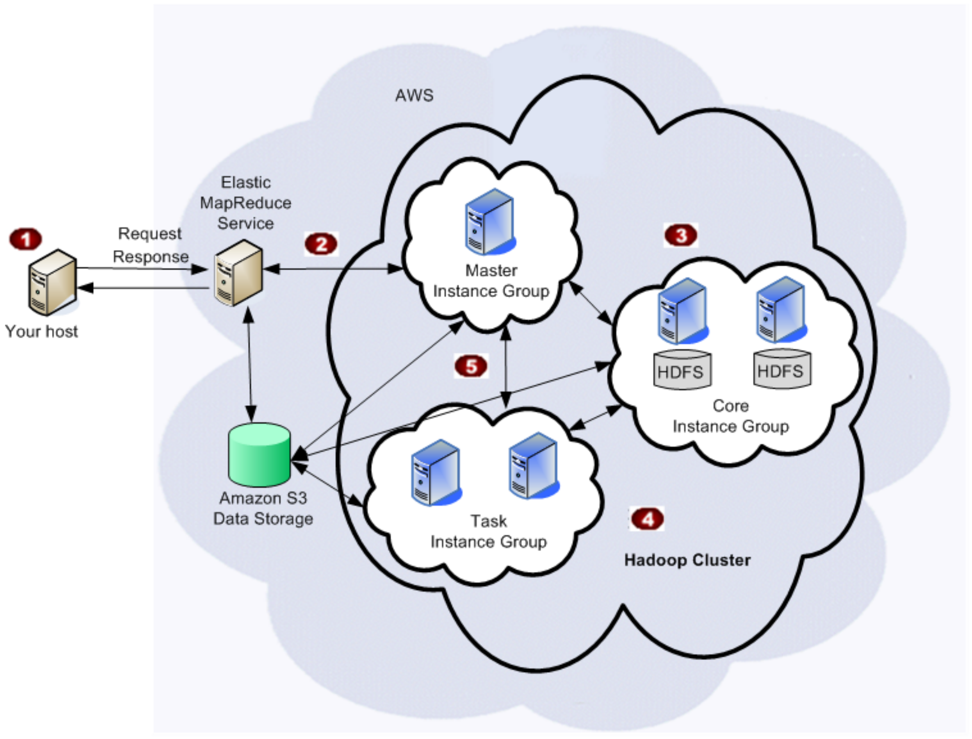
It is relying on Amazon EC2 for the creation of a Hadoop virtual cluster and on S3 for storing application input, output and configuration scripts.

The Hadoop cluster created by Amazon EMR consists from three different types of computing nodes:

- Master node - unique, part of the Master Instance Group, it controls the entire cluster by running the JobTracker daemon for schduling MapReduce tasks and the Namenode daemon for managing HDFS
- Core nodes - running the TaskTracker and Datanode daemons, thus, responsible with the MapReduce tasks execution and files storage in HDFS. Part of Core Instance Group, these nodes cannot be removed from the cluster because it could lead to HDFS failures
- Task nodes - running the TaskTracker daemon, they execute MapReduce tasks only. The Task Instance Group can be resized anytime during jobflow execution. The removal of some of these nodes or all of them will not lead to failures, Hadoop being in charge with the scheduling re-execution of tasks that were previously executed on the Task nodes.

By default, the Hadoop cluster consists from the Master node and several Core nodes.

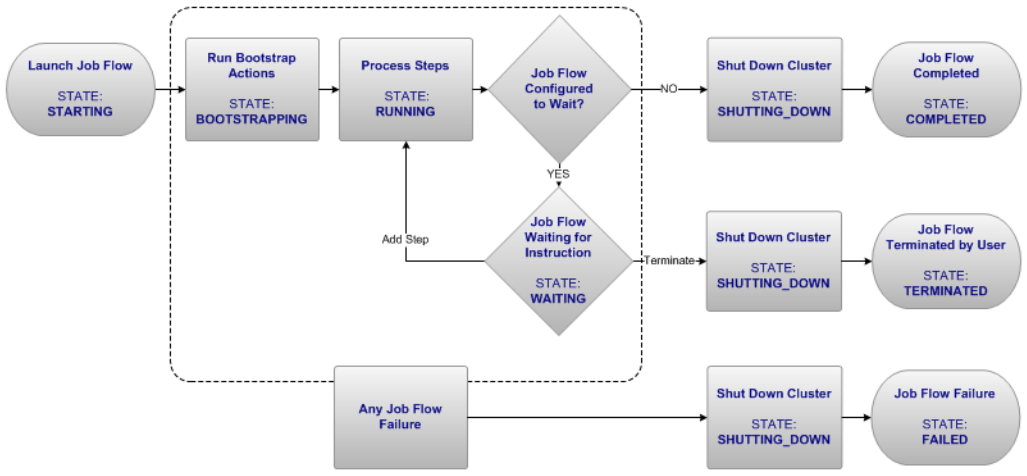
The architecture of Amazon EMR and the workflow of the jobflow are presented in the image below.



The data processing is represented by a list of instructions named steps that form a jobflow. A jobflow is executed on a dedicated Hadoop virtual cluster in the following way:

- the client uploads input data and executables to Amazon S3 and then it makes an EMR request that fires the jobflow execution
- EMR starts the virtual cluster and then runs the configuration scripts for Hadoop on each node
- Hadoop downloads the input data from Amazon S3 and executes the jobflow steps in the order they are defined
- Hadoop executes the jobflow and uploads the output to Amazon S3
- the jobflow being complete,the cluster is terminated and users can retrieve the jobflow output from S3

During jobfow execution, users can track its status by checking its state.The lifecycle of a jobflow is represented in the following diagram:



The jobflow is in STARTING state from the creation until the cluster's instances are up and running. Then the EMR will run the bootstrap actions on each cluster node, setting the jobflow's state to BOOTSTRAPPING. When bootstrapping phase is done, the cluster is ready to run the MapReduce applications thus the jobflow goes to RUNNING state. All the jobflow's steps are executed sequentially during this phase.

Once the steps execution is finished, the jobflow will go to the SHUTDOWN state unless it is configured to wait. In this case, it will go into the

WAITING state and wait until new steps are added to the jobflow or termination is requested by the user. During SHUTDOWN state , the Hadoop cluster is teminated and the service will set the jobflow state coresponding to the jobflow execution status.

There are several types of jobs supported by Amazon EMR:

- custom JAR - user must know Java and MapReduce API; he must convert the problem to map reduce tasks and implement the coresponding functions in the jar
- Hadoop streaming - user must provide the mapper and reducer in a programming language of his choice; They are executed like a standard custom jar by running the Hadoop streaming utility jar with them passed as parameters
- Cascading - executing complex, scale-free, and fault tolerant data processing workflows on a Hadoop cluster.Multitool is a command line application making it easy to execute Cascading workflows on Amazon Elastic MapReduce.
- Hive jobs - executes SQL-like queries over big database tables, as MapReduce jobs
- Pig - executes a Pig script

Main actions supported by Amazon EMR:

- RunJobFlow - for requesting the creation of a jobflow with defined steps that should run on an user defined number of virtual machines of a certain type
- DescribeJobFlows - allows the user to check the progress of the jobflow. This request will return a description of the jobflow
- AddJobFlowSteps - for appending new steps to the jobflow
- AddInstanceGroup - for adding new node types to the cluster
- ModifyInstanceGroup - for adding or removing nodes from a group (except that the Master Instance Group cannot be resized and the Core Instance Group cannot be reduced)
- TerminateJobFlow - for manually terminate a jobflow

Chapter 4

Problem statement

Besides the great advantages that Amazon EMR offers for large scale data processing, there are some serious limitations and restrictions that users meet.

An important restriction is the use of resources provided only by Amazon datacenters. Users that have access to scientific or private clouds are not able to use resources from them which could be less expensive or even free of charge. A restriction imposed by Amazon such that the demand won't exceed the number of resources from a datacenter is the limitation of instances started per account.

Another limitation is the lack of flexibility in operating systems/ Hadoop versions choice. Amazon is providing virtual machine images based on Debian Lenny and it uses older versions of Hadoop(0.18 and 0.20). Another case is when users have to process private data, they are not allowed to share it with external clouds. Another important disadvantage of Amazon EMR is the hourly fee that users have to pay additional to one of Amazon EC2 and the extra-cost for data transfer to and from S3. The cost for an Amazon EC2 instance is rounded up to the hour, the instance usage for 1 minute has the same price with its usage for one hour.

Related work has been focused on MapReduce adaptation to the cloud properties. This was done by modifying existing systems leveraging Amazon services or creating new implementations of MapReduce on top of existing platforms as Microsoft Azure platform.

The goal of this thesis is the implementation of an MapReduce execution platform that uses resources from multiple clouds. Besides the functionalities of Amazon EMR, the service is allowing clients to use different virtual machine images or Hadoop versions and it is free of charge for the users that have access to private or scientific clouds (academia staff).

Another important aspect is that the service is able to use different types of Iaas clouds that are implementing the EC2 API for managing computing resources. Open-source versions of cloud computing architectures used by our service are Nimbus, OpenNebula and Eucalyptus.Furthermore, it can use S3 compatible storage systems as Cumulus or Walrus.

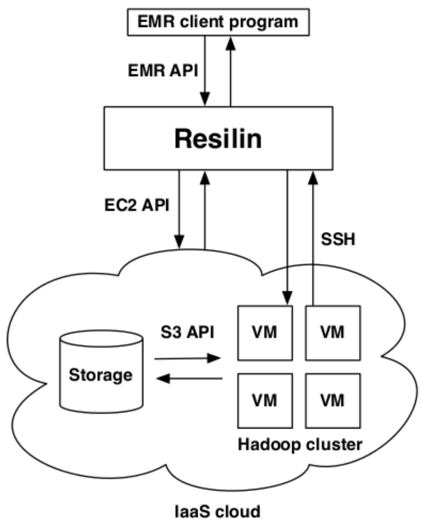
Chapter 5

Architecture and implementation

As Amazon EMR, the service implemented for this thesis, Resilin, is providing an abstract layer between the IaaS cloud and the client program. It implements the main functionalities of Amazon EMR and also some important improvements over it, the major one being the usage of resources from multiple clouds(even different types of clouds).

5.1 Arhitectural overview of Resilin

Resilin architecture is presented in the figure below. As it can be noticed, there are three entities involved in the processing:the client program, Resilin and the Iaas cloud.



The client program will interact with the service through EMR requests. Resilin receives the client requests and translates them in two types of actions: EC2 requests to the IaaS cloud for starting or terminating instances and remote connections using SSH protocol to the instances for Hadoop configurations.

The steps of running a jobflow for MapReduce execution using Resilin are the following:

- client uploads input, executables and bootstrap scripts to the S3 compatible storage used by the cloud
- client makes a RunJobFlow request to the service
- Resilin validates the parameters of the request and makes an EC2 request for starting instances
- Resilin configures the running instances through SSH
- Resilin starts jobflow steps execution on the master instance
- Resilin terminates the virtual cluster when jobflow execution ends

During jobflow execution, the client program is able to retrieve jobflow status(DescribeJobFlows), add new steps to the computation(AddJobFlowSteps),add new instance group to the Hadoop cluster(AddInstanceGroup), increase/decrease the number of instances in a group (ModifyInstanceGroup) and terminate the jobflow (TerminateJobFlow).

The implementation was done in Python, using the boto library for interaction with clouds (EMR and EC2 API). For the SSH connections,it uses the paramiko library, and for the http requests and response the Twisted framework.

5.2 Using Resilin

Before making requests to the service, the client must load the virtual machine image used for instance creation (AMI), the input data, bootstrap scripts and executables to the cloud storage (eg. Cumulus). This process has no interaction with Resilin, it depends only on the client program.

5.2.1 Creating a Jobflow

For creating a jobflow in Resilin and starting a virtual Hadoop cluster, the client program must make a RunJobFlow request with the following format:

```
https://elasticmapreduce.amazonaws.com?Operation=RunJobFlow
&Name=MyJobFlowName
&Instances.MasterInstanceType=m1.small@paramount-15.rennes.grid5000.fr
&Instances.SlaveInstanceType=m1.small@paramount-15.rennes.grid5000.fr
&Instances.InstanceCount=4
&Instances.Ec2KeyName=myec2keyname
&Instances.KeepJobFlowAliveWhenNoSteps=true
 ...
&AuthParams
```

As it can be noticed in the example above, there are some important parameters specified into the request that are defining the Hadoop cluster needed for job execution:

- instance type - the difference from Amazon is that, for Resilin, besides the type of the instance(m1.small, m1.medium, etc.), the client must provide also the frontend address of the cloud where the machines are going to be started.

  Eg. Amazon : "m1.small"

  Resilin: "m1.small@paramount-15.rennes.grid5000.fr"

- InstanceCount - representing number of instances to start in the Hadoop cluster

For each step in the jobflow the location of the executable and the arguments must be provided, Resilin will download it from the storage system to the Hadoop master node.

Currently, Resilin supports only two types of jobflow steps:

1. Custom JAR - where the Hadoop MapReduce application is implemented as a Java jar. The coresponding parameters that must be added to the RunJobFlow request for this type of step are the following:

```
 ...
&Steps.member.1.Name=MyStepName
&Steps.member.1.ActionOnFailure=CONTINUE
&Steps.member.1.HadoopJarStep.Jar=cumulus://mybucket/jar
&Steps.member.1.HadoopJarStep.Args.member.1=cumulus://mybucket/input
&Steps.member.1.HadoopJarStep.Args.member.2=cumulus://mybucket/output
 ...
```

2. Streaming - where the client provides the source for single-step mapper and reducer functions (languages other than Java can be used for this).

```
 ...
&Steps.member.1.Name=MyStepName
&Steps.member.1.ActionOnFailure=CANCEL_AND_WAIT
&Steps.member.1.HadoopJarStep.Args.member.1="-mapper"
&Steps.member.1.HadoopJarStep.Args.member.2="mapper_file_path"
&Steps.member.1.HadoopJarStep.Args.member.3="-reducer"
&Steps.member.1.HadoopJarStep.Args.member.4= "reducer_file_path"
&Steps.member.1.HadoopJarStep.Args.member.5="-input"
&Steps.member.1.HadoopJarStep.Args.member.6="cumulus://mybucket/input"
&Steps.member.1.HadoopJarStep.Args.member.7="-output"
&Steps.member.1.HadoopJarStep.Args.member.8="cumulus://mybucket/output"
&Steps.member.1.HadoopJarStep.Jar=~hadoop/contrib/streaming/hadoop-streaming.jar
 ...
```

After receiving the RunJobFlow request, Resilin validates the parameters and creates the jobflow. The response sent back to the client consists of an xml containing the jobflowID (parameter used later for requests that modifies jobflow components).

Cluster deployment

First step in jobflow execution is the provisioning of a Hadoop virtual cluster. This is achieved by making EC2 requests to the Iaas cloud to start the requiered number of instances.

Resilin makes two EC2 requests to the Iaas, first request is for starting one instance which will be the Master of the cluster and another request for starting the core nodes[1].

When the instances are running, the jobflow will go from *STARTING* into *BOOTSTRAPPING* phase. At this step, Resilin connects to the machines using SSH and configures them by setting default Hadoop properties(HDFS Namenode address, MapReduce JobTracker address), starting Hadoop daemons on the machines, preparing HDFS filesystem and running client bootstrap actions.

For bootstrapp phase, on each machine in the cluster, the service is storing a file with the instance and jobflow properties. This way, scripts or commands can be executed on certain machine types.

When all settings are done, the Hadoop cluster is ready for MapReduce jobs execution and the jobflow will change its state from *BOOTSTRAPPING* to *RUNNING*.

Jobflow steps execution

A step of the jobflow refers to the execution of a MapReduce application. When jobflow goes into *RUNNING* state, it starts to process all the steps sequentially.

For starting the execution of a step, Resilin downloads the executable on the master instance and runs the command /home/hadoop/hadoop jar <executable> <args> for starting Hadoop processing.

If during the execution of the step, an exception occurs, then, depending of the *ActionOnFailure* property of the step, the jobflow will abort or continue to execute.

After all the steps are executed, Resilin will shutdown the cluster or can set it to wait for an user action (add new steps).

[1]If InstanceCount parameter is 1,then, the same virtual machine performs the roles of master node and core node.

5.2.2 View Jobflows Details

For monitoring the status and retrieve extended information about a jobflow execution, the client can make DescribeJobFlows request passing the jobflow ID as parameter.

Sample request:

```
https://elasticmapreduce.amazonaws.com?Operation=DescribeJobFlows
&JobFlowIds.member.1=j-3UN6WX5RRO2AG
&AuthParams
```

Resilin searches for the jobflow with the specified ID in the user's jobflows list and, if there's a match, it will store all the jobflow properties in an xml. The xml will be sent as a response to the client. The client can make DescribeJobflows request any time during the jobflow execution.

5.2.3 Terminate a Jobflow

When a jobflow execution is finished and the jobflow is not set to wait, the TerminateJobflow action is automatically called. This will close the connections to the machines and will shut down the cluster. For jobflows that are in the *Waiting* state, the client must make a request for jobflow termination. This request must be made when there is no computation to execute, otherwise, resources are wasted with no purpose. A jobflow can be termminated any time during its execution.

5.2.4 Hadoop cluster resizing

One of the most important features of Amazon EMR and Resilin is the scalability of computing power and resources number. On-Demand, new resources can be added or removed from the computation.

This thesis focuses more on the improvement of this functionality. While Amazon EMR uses resources from one of its datacenters for scaling up or down the computing resources, Resilin can use resources from different clouds, of different types.

It can use resources from multiple clouds where the client has access. For example, if there are two clouds A and B, the client started a jobflow that uses resources from A, then he is also able to add some new resources to the jobflow by requesting to add new machines from B.

The two actions from the EMR API that involves the computing resources scalability are the *AddInstanceGroups* and *ModifyInstanceGroups*.

Adding instance groups

This action is scaling up the computing resources number. It is adding a new group of instances to the Hadoop cluster (CORE or TASK). By default, there is a Master instance group, thus, only CORE and TASK groups can be added.

Request format:

```
https://elasticmapreduce.amazonaws.com?Operation=AddInstanceGroups
&JobFlowId=j-3UN6WX5RRO2AG
&InstanceGroups.member.1.Name="Task Instance Group"
&InstanceGroups.member.1.InstanceRole=TASK
&InstanceGroups.member.1.InstanceType=m1.small@cloudB
&InstanceGroups.member.1.InstanceRequestCount=2
&AuthParams
```

The response to this request will return the id of the newly created group. The client must provide the jobflow id whose resources he wants to increase, with the machine's type, number and role(core or task). The type is specifying the cloud address where the client wants to start the machines.

After the request parameters validation, Resilin will contact the cloud specified in the type (cloudB) and will request the launching of two new machines on it. When the machines are running, it will make the default Hadoop configurations and starts the daemons coresponding to the role of the instance. Then the machines will contact the master instance from cloudA and ask for tasks to process, thus joining the Hadoop cluster.

Modify instance groups

Unlike *AddInstanceGroups*, this action allows scaling up and down an existent group of instances. The parameters provided to the request must include the group id and the number of resources it should have. Request format:

```
https://elasticmapreduce.amazonaws.com?Operation=ModifyInstanceGroups
&InstanceGroups.member.1.InstanceGroupId=i-3UN6WX5RRO2AG
&InstanceGroups.member.1.InstanceRequestCount=2
&AuthParams
```

When receiving the request, Resilin compares the requested number of instances and the current one. If the current one is smaller, then it will make a request to the cloud for starting new machines, else, if the current number is greater then it will terminate the last machines added to the group.

Chapter 6

Evaluation

For the evaluation of our implementation we ran various tests on both Amazon EMR and Resilin. Resilin was used on top of Nimbus clouds, using Cumulus (which is provided with Nimbus) as a storage system.

Testing configurations :

System	Amazon EMR	Resilin
IaaS	Amazon EC2	Nimbus
Storage system	Amazon S3	Cumulus
Hadoop version	customized 0.20.0	0.20.2
Instance	c1.medium(2.13 GHz - 2.33 GHz)	c1.medium(2.5 GHz)

The evaluation consists of two parts: analysis of the cluster deployment time and execution time for two MapReduce applications. Experiments were made using different cluster sizes.

6.1 Deployment Time

The cluster deployment time represents the time interval between jobflow submission(RunJobFlow request) and steps execution start. It corresponds to the Hadoop cluster provisioning and machines configuration.

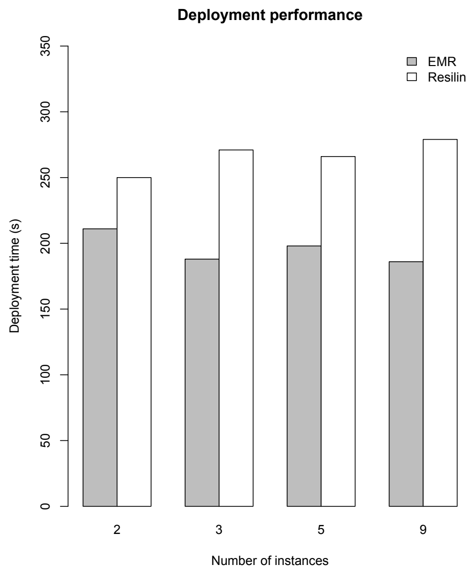
The cluster provisioning includes the propagation of virtual machine image from the cloud storage to the hypervisor nodes and the start of instances.

This phase is indepedent of the MapReduce application submited. It is only influenced by the number of machines requiered to start. As the number increases, the time taken for starting machines and configure them increases too.

Another factor weighting in this experiment is the size of virtual machine image. If the image is large, it will take more time to transfer it from the cloud storage to the hypervisor nodes.

The deployment time was computed as the the difference between the jobflow CreationDateTime and ReadyDateTime reported using DescribeJobFlow action.

The experiments were run using different number of instances, the results obtained are shown in the image below.



Both Amazon EMR and Resilin have a stable deployment time. The measurements show that Resilin is constantly slower than Amazon. As the Amazon EC2 architecture is not public, we don't know how they are propragating the virtual machine image or if they keep it in a cache on the hipervisor nodes.

Improvements that could be done on Resilin for reducing the deployment time are the image compression or caching it on hypervisor nodes. Also we can reduce the SSH connections during configuration(currently there is made a connection per configuration).

6.2 Execution Time

The execution time corresponds to the time interval between Jobflow StartDateTime and EndDateTime. For this experiments, we ran two jobflows consisting from one step each. The execution time was computed as the difference between the StartDateTime and EndDateTime step properties.

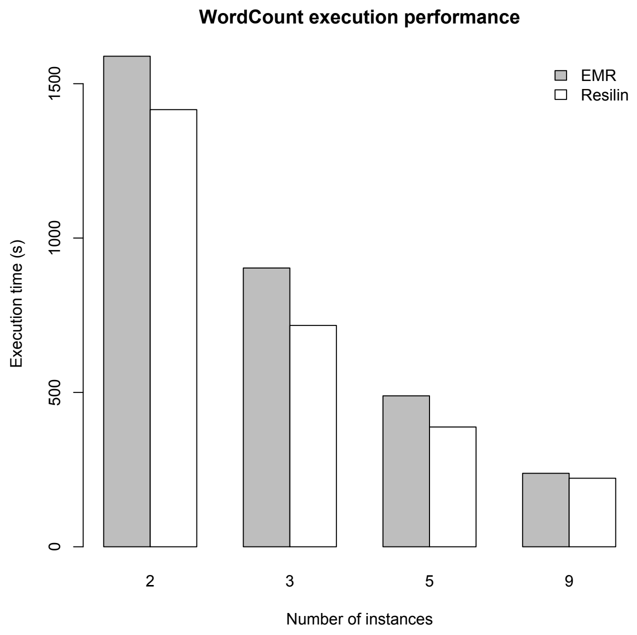
Similar Hadoop configurations were used on both Amazon EMR and Resilin. Two cores virtual machines were used for instances. During processing, Hadoop was running 2 map tasks per core and 1 reduce task.

1. Wordcount

   This program reads text files and counts the number of occurences of words. The mapper takes one line of text and splits it into words emitting a key/value pair (word,1).The reducer will combine the map output and add the values for a word returning the number of occurences of that word.

   We ran a streaming jobflow, providing the mapper and reducer codes written in python. The input was represented by text files summing 1.8 GB. As discussed earlier, each slave node ran 4 map tasks in parallel and 2 reduce tasks.

   The results obtained are displayed in the next figure:



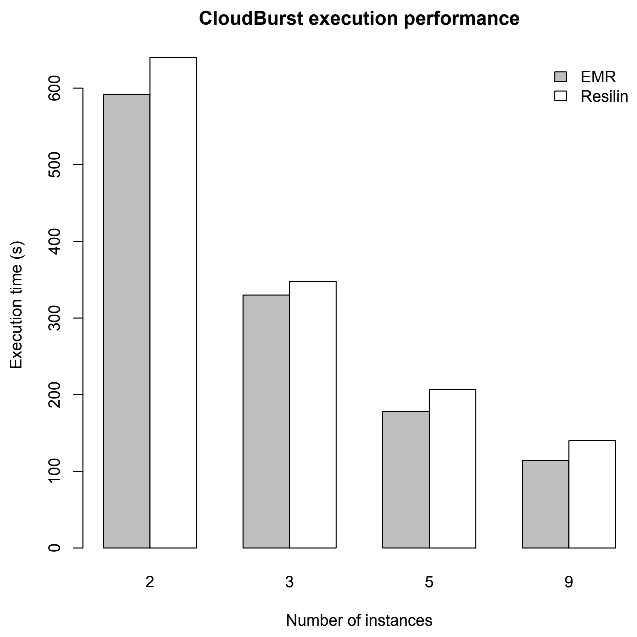
   Resilin was faster in all the cases(the cpu frequency of Resilin instances was higher than Amazon's) but as it can observed, the time difference is decreasing with the addition of machines. This may be due to the

Cumulus storage that resides on a single machine(the service node of Nimbus).

2. Cloudburst

   This application is mapping next-generation sequence data to the human genome and other reference genomes.For its execution we used a custom JAR jobflow with the parameters specified in Amazon EMR documentation(240 map tasks and 48 reduce tasks).

   The figure below shows the results:



   In this case, Resilin is slower than Amazon EMR. Since Amazon uses a modified version of Hadoop, we don't know exactly if the performance difference is due to execution environment customization.

For this experiments,Resilin was used with Nimbus clouds deployed on Grid5000. It showed a similar level of performance with Amazon EMR.

Chapter 7

Conclusion

The goal of this thesis was the implementation of a service that should provide the same functionalities as Amazon Elastic MapReduce but on top of multiple clouds. This will enable users with access to private or scientific clouds to run Mapreduce computations without having to worry about cloud resources management, failures handling,etc.

Resilin is using resources from clouds deployed with open-source Iaas toolkits that implement the EC2 interface (Nimbus, OpenNebula, Eucalyptus).

From the test results analysis, we can say that Resilin offers the same level of performance as Amazon EMR, in the same time adding important improvements to its functionalities.

Future Work

There are some features left to implement for a complete compatibility with Amazon EMR API. One of these is to offer support for the execution of other types of jobflows: Cascading, Hive, Pig.

Improvements could be done on the current implementation also. We need to make a deeper deployment process analysis for finding the reasons why Resilin is slower than Amazon EMR. Another improvement could be done on the cloud selection, currently, users must provide the cloud address where the virtual machines are going to be deployed.

There must be done some tests for checking the functionality of Resilin when using OpenNebula and Eucalyptus clouds (we mentioned before that for implementation and evaluation Resilin used only Nimbus clouds).

Bibliography

[1] Amazon Elastic MapReduce, *http://aws.amazon.com/elasticmapreduce/*

[2] Jeffrey Dean,Sanjay Ghemawat *MapReduce: Simplified Data Processing on Large Clusters*

[3] Tom White, *Hadoop - A Definitive Guide*

[4] Qi Zhang, Lu Cheng, Raouf Boutaba, *Cloud computing: state-of-the-art and research challenges*

[5] James Murty, *Programming Amazon Web Services*

[6] Apache Hadoop, *http://hadoop.apache.org/*

[7] Nimbus, *http://www.nimbusproject.org/*

[8] paramiko, *http://www.lag.net/paramiko/*

[9] boto: A Python interface to Amazon Web Services *http://boto.cloudhackers.com/*

[10] T. Gunarathne,Wu, J. Qiu and G. Fox, *MapReduce in the Clouds for Science*

[11] H. Liu, *Cutting MapReduce Cost with Spot Market*