

ALMA MATER STUDIORUM · UNIVERSITÀ DI  
BOLOGNA

---

SCUOLA DI SCIENZE  
Corso di Laurea in Informatica

# Big Data Analytics and Application Deployment on Cloud Infrastructure

Relatore:  
Chiar.mo Prof.  
Gianluigi Zavattaro

Presentata da:  
Iacopo Talevi

II Sessione  
Anno Accademico 2016/2017



# Introduction

The work related to this project began in October 2016. It was born from the collaboration between Mr. Alessandro Bandini and me, and has been developed under the supervision of professor Gianluigi Zavattaro. The main objective was to study, and in particular to experiment with, the cloud computing in general and its potentiality in the data elaboration field.

Initial guidelines were based on a *University of Pennsylvania* course. The first goal was to study and delve into cloud technologies and the cloud world in general. This was necessary because cloud computing is not covered in the first three years of our university degree. We searched various materials such as books, scientific papers, online courses, websites, practice guides and anything that could be useful. After this screening operation, we selected the most important materials for our project and we began to study them in more details.

Once we had acquired the basic knowledge, we tried to always combine the theoretical study with exemplifying practical experiences. For example we found a very useful online course called “Intro to Hadoop and MapReduce” that offered after each lesson some exercises or experiments to test and to put into practise the learned knowledges through virtual machines.

We extended our experiments using real cloud platforms too, such as *Amazon Web Service (AWS)*. During the project we had to change platform and the final version is realized through *Google Cloud Platform*. We used a collaboration between our university and *Injenia Group* that offered us free credits to work on our project within Google’s platform.

We worked on this project also during our internal internship. With the Professor, we identified the development of a Social Network integrated with cloud data elaboration to offer useful integrated information as our final objective. In particular we wanted to develop a University Social Network specific for Bologna University.

The Social Network was initially developed using the Amazon services but after a small problem with our free credits we decided to move to a IaaS solution to avoid *vendor lock-in* problems and to obtain a more independent solution.

Data elaboration is performed through *MapReduce* applications written in Python and executed using *Google App Engine*. MapReduce is a programming approach for solving big data analysis problems. Alessandro developed the same applications also through a functional approach with the *Scala* language and *Spark* platform, to compare the produced code.

Alessandro and I collaborated during the entire project, working side by side on every aspect, but in general I focused my work on the Social Network realization, with all aspects connected to the development of the interface and the infrastructure configuration while Alessandro focused on the data elaboration. For this reason, some aspects of the project will only be mentioned in this dissertation because they have already been covered in details in Alessandro's work.

This dissertation starts with a theoretical introduction on cloud computing, analyzing the main aspects, the keywords, and the technologies behind clouds, as well as the reasons for the success of this technology and its problems.

After the introduction section, I will describe the three main cloud platforms in the market. We directly used two of them.

Subsequently I am going to describe the main aspects of our practical project. I will analyze the social network development, with the initial solution realized through Amazon Web Services and the steps we took to obtain the final version with its characteristics.

To conclude, the last section is specific for the data elaboration and contains a initial theoretical part that describes *MapReduce* and *Hadoop* followed by a description of our analysis. I will explain the basic idea, the code and the problems encountered.



# Contents

<b>Introduction</b>	<b>i</b>
<b>1 Cloud Computing Introduction</b>	<b>1</b>
1.1 General Introduction . . . . .	1
1.2 Technologies behind Cloud Computing . . . . .	2
1.3 Cloud Keywords . . . . .	3
1.4 Cloud Computing Architecture . . . . .	4
1.5 Cloud Computing Models . . . . .	5
1.5.1 Public Cloud . . . . .	5
1.5.2 Private Cloud . . . . .	6
1.5.3 Hybrid Cloud . . . . .	7
1.5.4 Community Cloud . . . . .	7
1.6 Cloud Services Reference Models . . . . .	8
1.6.1 Infrastructure-as-a-Service (IaaS) . . . . .	9
1.6.2 Platform-as-a-Service (PaaS) . . . . .	10
1.6.3 Software-as-a-Service (SaaS) . . . . .	11
1.6.4 Summary . . . . .	13
1.7 Service-and-Compliance-level Agreements . . . . .	13
1.8 Cloud Economic Aspects . . . . .	13
1.8.1 Economic Advantages . . . . .	14
1.8.2 Pricing Strategies . . . . .	14
1.9 Reasons of Cloud Success . . . . .	15
1.10 Obstacles and major Challenges of Cloud Computing . . . . .	16

---

1.11	Cloud Vulnerabilities . . . . .	18
1.12	Legal Issues of Cloud Computing . . . . .	21
1.13	Energy Use and Ecological Footprint . . . . .	22
1.14	Cloud Applications . . . . .	22
<b>2</b>	<b>Cloud Platforms</b>	<b>25</b>
2.1	Cloud computing at Amazon . . . . .	25
2.1.1	EC2 and Connected Services . . . . .	26
2.1.2	Storage Solutions . . . . .	28
2.1.3	Useful Services . . . . .	30
2.2	Cloud Computing: Google Perspective . . . . .	32
2.2.1	Google AppEngine . . . . .	32
2.2.2	Storage Solutions . . . . .	34
2.2.3	Useful Services . . . . .	34
2.2.4	Develop, Deploy and Maintain process . . . . .	35
2.3	Microsoft Windows Azure . . . . .	36
2.3.1	Compute Services . . . . .	37
2.3.2	Storage Solutions . . . . .	40
2.3.3	Azure Peculiarity . . . . .	41
2.4	Other Possibilities . . . . .	41
<b>3</b>	<b>University Social Network</b>	<b>43</b>
3.1	Interface . . . . .	44
3.2	First Attempt using AWS . . . . .	46
3.2.1	EC2 . . . . .	46
3.2.2	DataBase . . . . .	47
3.2.3	Amazon Elastic MapReduce . . . . .	48
3.2.4	The end of the Amazon Experience . . . . .	49
3.3	Application Development using our University Machines . . . . .	49
3.3.1	Cassandra . . . . .	49
3.3.2	Test Phase . . . . .	50
3.4	Google Solution . . . . .	50



---

3.4.1	Machines Characteristics . . . . .	51
3.4.2	Packages Installation . . . . .	52
3.5	Conclusion . . . . .	53
<b>4</b>	<b>Data Elaboration</b>	<b>55</b>
4.1	MapReduce . . . . .	56
4.1.1	MapReduce Data Flow . . . . .	57
4.1.2	MapReduce Features . . . . .	58
4.1.3	Examples . . . . .	58
4.2	Hadoop . . . . .	60
4.2.1	Comparison between Hadoop and other alternative systems . . . . .	60
4.2.2	Hadoop Distributed File System (HDFS) . . . . .	61
4.2.3	Hadoop Data Flow . . . . .	61
4.3	Analysis using Google App Engine . . . . .	62
4.3.1	Job Types . . . . .	63
4.3.2	Stages . . . . .	63
4.3.3	Input Reader and Output Writer . . . . .	64
4.3.4	Configuration Settings . . . . .	65
4.4	MapReduce Pipeline example in Python . . . . .	66
4.5	Our Elaboration . . . . .	68
4.5.1	Social Ranking . . . . .	69
4.5.2	Movie Ranking . . . . .	74
4.5.3	Movie Genres Ranking and Movie Suggestion . . . . .	78
4.5.4	Performance Analysis . . . . .	79
4.6	Data Elaboration Conclusion . . . . .	82
	<b>Conclusions</b>	<b>83</b>
	<b>A Code</b>	<b>85</b>
A.0.1	Pipeline Adapter Classes . . . . .	85
A.0.2	Map Task to generate the result on a single file . . . . .	86

A.0.3	Map Task to sort the result and return it in JSON format	86
A.0.4	Movie Genres Ranking . . . . .	87
A.0.5	Movie Suggestion . . . . .	90
	<b>Bibliography</b>	<b>97</b>

# List of Figures

1.1	Cloud Computing Architecture. Source: Mastering Cloud Computing Foundations and Applications Programming [1] . . . . .	4
1.2	Cloud Computing Reference Models. Source: Mastering Cloud Computing Foundations and Applications Programming [1] . . . . .	9
3.1	SocialUNIBO homepage . . . . .	45
3.2	Advice Page . . . . .	46
3.3	Compute Engine instance creation form . . . . .	51
4.1	Total computation time with a different shard number . . . . .	81



# List of Tables

4.1	Execution time depending on the shard number . . . . .	80
4.2	Average map calls per second depending on the shard number	81



# Chapter 1

## Cloud Computing Introduction

This chapter will describe the basic knowledges about cloud computing that Alessandro and me learned on the first part of our project. We dedicated about three months, from November to January, to search and study various materials from different sources. These knowledges that cover the principal aspects of cloud computing has been fundamental during all phases of our project to understand the most difficult concepts and the practical parts.

### 1.1 General Introduction

“Cloud computing is a utility-oriented and Internet-centric way of delivering IT services on demand. These services cover the entire computing stack: from the hardware infrastructure packaged as a set of virtual machines to software services such as development platforms and distributed applications.”[1]

The cloud originally was the Internet symbol in the network system diagrams, but now this term is used to represent both components of the new cloud computing system. In fact it is used to refers to the available applications deployed on the cloud and to the hardware and software infrastructure in the datacenters that compose and create the cloud.

**Basic Idea** Cloud computing is based on the idea to supply IT infrastructure like a public utility through a central system view where the hardware is located in large datacenters. This idea was considered realistic only during the last ten years of the 20th century thanks to the developments of infrastructure and the Internet that allow a fast and cheap user connection with datacenters. Cloud computing is based on the client-server paradigm usually with stateless servers. These characteristics are preferred because the obtained systems are simpler, more scalable and robust.

**The origin of Cloud Computing** Cloud computing was made possible only thanks to the technology development in the network infrastructure and in the parallel and distributed computing field. In the beginning of the 2000s all the big IT companies began to implement their cloud infrastructure. The first was *Amazon* that released a first version of cloud computing service called *EC2* in 2006. In the following years *Google*, *Microsoft*, *Apple*, *Oracle* and others entered this market.

**Cloud Computing Revolution** In the last years the developers' way to think and realize applications and computing systems in general has been radically changing connected with the cloud computing development. The idea of *everything as a service (XaaS)*, very common today, was borne with the cloud computing where, as mentioned above, all the components of the computing stacks can be requested and rented like a service.

## 1.2 Technologies behind Cloud Computing

There are some essential technologies that enabled the rise of cloud computing:

**Distributed System** is the base of the cloud infrastructure. In fact this is a collection of an enormous amount of independent systems that handle



and share the workload. They appear to the user like a single system with a single interface.

**Virtualization** allows the best usage of very powerful centralized datacenters. In fact for example hardware virtualization allows deploying independent virtual machine on the same hardware. So a single powerful server can be used at the same time by a lot of different users. This technology has also other advantages like the possibility to dynamic change the characteristics of each virtual machine and the possibility to move these on different hardware at anytime.

**Web 2.0** introduced interactive web applications that can substitute the normal desktop versions with various advantages. All the software offered through the cloud are based on this technology.

## 1.3 Cloud Keywords

Cloud computing can have different shapes but it is possible to find some common keywords.

**On demand** For the end users the services are always accessible from anywhere, they need only a Internet connection.

**Reliability** The ability of cloud systems to perform its required functions under stated conditions.

**Scalability / Elasticity** gives the possibility to adapt the infrastructure to the business needs and the peak workload in any moments on-demand. The system can add new components (*Horizontal Scalability*) or improves the existing nodes (*Vertical Scalability*). The scalability concept can be applied across the entire computing stack through cloud computing, from the hardware components (like compute capability, storage and networking) to the services and application components that can be used and requested on demand.

**Virtualization** the cloud infrastructure are built through this technology

**Cost Model** all cloud forms implement a utility-based cost model (or pay-per-use strategy), where the user pays only what he really uses.

## 1.4 Cloud Computing Architecture

A cloud physical infrastructure is usually composed by more than one datacenter and it can be heterogeneous. Depending on the services offered, different software layers can be used on the physical infrastructure. Usually there is a component to create and manage the virtual machines and if necessary a development platform or directly the applications that users can use.

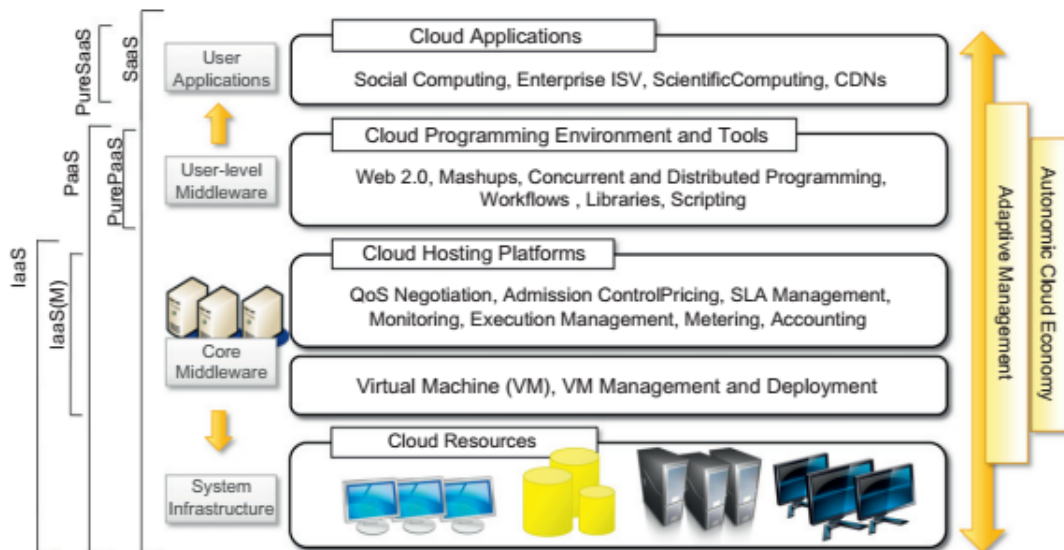


Figure 1.1: Cloud Computing Architecture.

Source: Mastering Cloud Computing Foundations and Applications Programming [1]

Different virtualization techniques are used to create the virtual machines, to expose the complete infrastructure as a collection of virtual machines and

to create a runtime environment where users can deploy and manage their applications.

## 1.5 Cloud Computing Models

The term *Cloud* refers to the infrastructure through which several different services are offered to the users. Depending on who control the infrastructure (*the administrative domain*) and who can use the services there are different deployment models.

### 1.5.1 Public Cloud

is the most common and famous cloud computing deployment model. It is characterized by a company that sells cloud services to the general public on a subscription basis. All customers who own a credit card to pay the provider can use the services. They can access the cloud from anywhere and at any time, this action requires only an Internet connection.

The cloud provider implements the service through a series of interconnected datacenters. These datacenters are spread all over the world to offer users good performances everywhere. They use the virtualization technology to serve a large amount of customers with a smaller number of devices, offering to each client a private and isolate computing environment. This situation is called *multitenancy*.

In the last years, many small companies, in particular start-ups, used the public cloud to obtain the IT resources that they need. This solution has various advantages: they do not have to make a big initial investment to build a private IT infrastructure, do not need to create immediately a complete IT department that manage the infrastructure, do not have to predict the resource that they will need in the future and they do not have to design a final infrastructure because the cloud offers the possibility to adapt his services with the business needs in any moment. Public cloud is also used to extend a company IT infrastructure to serve momentaneous or particular

needs.

In the public cloud it is possible to find all type of cloud services: basic IT infrastructure, of which the most famous provider is Amazon with *Amazon EC2*, as well as platform services, a field in which Google is the top vendor with *Google AppEngine*, and there are a very large number of cloud applications vendors providing all types of possible applications.

### 1.5.2 Private Cloud

Public Cloud has a lot of advantages for companies but it has also some drawbacks. In particular these depend on the fact that the physical infrastructure is managed by the cloud provider so an external third-party company. This situation could create problems to ensure the secrecy of confidential information or other problems connected with the loss of control on the system. Another aspect is that many companies, founded before the cloud computing spread, already own a IT infrastructure and they do not want to waste their investments. Rather, they may prefer to take advantages from the cloud but using their own infrastructure; the solution is the Private Cloud.

In this deployment model a company that own a large IT infrastructure implement on it a private cloud system. Only the organization members can use the offered services. They do not have to pay for the service but usually they have some limitation for each user or for each project/department. As mentioned above, this is a solution for companies that already have a big amount of IT resources or that want to have complete control on their system and in particular on their confidential information. The control of the system allows also the company to ensure a specific quality of service and to be sure that particular standards or procedures are followed during the system management or the application deployment.

There exist a lot of open source or property softwares to deploy a cloud system on a private infrastructure. The company needs one or more software layers depending on what types of service it wants to offer.

### 1.5.3 Hybrid Cloud

As mentioned above, public and private cloud have some advantages and some disadvantages. Public cloud offers huge amount of resources that enable users to handle peak workload without problems but it suffers from security threats and it does not give users a complete control of the system. Private cloud, on the contrary, guarantees security for confidential information and a complete control of the system but it has limited resources.

So the solution could be to compose the two models to take advantages from both. In fact sometimes companies have some needs like confidentiality that is unable to meet with a public cloud model. But they may also not own enough IT resources to develop a completely private solution, so they can compose the two models. They can use the positive aspect of each to serve the specific organization's needs and to cover the other model problems.

Usually this type of cloud is based on a private cloud that can use public cloud resources to manage workload peak or to increase the computing capability for a specific time period. This operation is called *cloudbursting*. In a hybrid solution public cloud services can be also directly integrated in the private cloud.

It is important for the company to have a good workload management system that minimize the public cloud resources used and that immediately release them when they are no more necessary, to optimize the costs. So the workload management system in a hybrid cloud is more complex than public or private cloud, this is also because it has to ensure that confidential information are processed only in the private system and that the public infrastructure perform only secondary operations with no secrecy problems.

### 1.5.4 Community Cloud

A community cloud is implemented through multiple infrastructure from different administrative domains that work together. This happens because in some occasions a group of companies or a community of people that have

the same needs decide to collaborate to create a cloud. They share their resources to create the best solutions for their common needs. It is different from public cloud where users with very different needs can use the same services. This infrastructure is controlled by the community, usually through democratic processes, and only the community members can use it.

Community cloud model identify also a situation where normal users share their underutilized resources to create the cloud infrastructure. In this situation each user is at the same time a consumer and a provider of the cloud services.

Community cloud has some positive aspects:

- the absence of a single services vendor allows a more flexible and open system
- all the users collaborate to create the infrastructure. So it is spread between the users and consequently there is not a single point of failure identified by the provider datacenter
- the users are consumers and providers at the same time so there is not a conflict between convenience and control
- this model is not realized through large polluting datacenters so it is also more eco-friendly.

## 1.6 Cloud Services Reference Models

The cloud services offered to the users can be divided in three major categories, called reference models: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). They offer different degrees of freedom because they are located at different levels of the software stack.

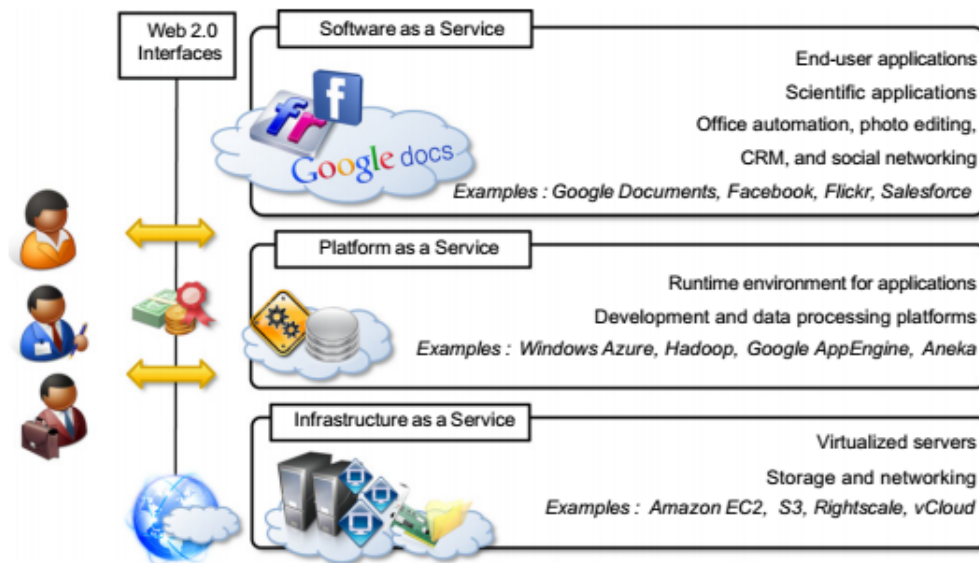


Figure 1.2: Cloud Computing Reference Models.

Source: Mastering Cloud Computing Foundations and Applications Programming [1]

### 1.6.1 Infrastructure-as-a-Service (IaaS)

is the most basic cloud service. Users request a basic computing infrastructure provided through virtual machines, created using hardware virtualization. The machines have variable values of processing capacity, storage and networking and they are priced depending on these values.

Users are totally independent about the softwares installed on the rented machines, they can choose all software components starting from the operating system. On the contrary they do not have the control of the physical infrastructure, the hardware components, that is handled by the provider. Users can request levels of IT resources, like processing, storage or networking and the provider manage the physical infrastructure to satisfy the demand. The hardware component are shared between the customers thanks to virtualization technology.

This model is usually used to recreate a pre-existing server environment on

the cloud or to create a new one as independent as possible from the cloud providers. It is the best solution for new companies that want a completely customizable IT infrastructure without a big initial investment and without software limitations.

The best advantage of IaaS with respect to a traditional private system, is the ability to dynamically adapt the resources used to the workload. So the over-provisioning problem is eliminated and the customers pay for the exact amount of resources that their systems used. This feature is very useful for systems with a volatile demand, because it allows to optimize the costs.

However, the high degrees of freedom implicate that the developers can not use the convenient high level services offered by the providers and they must have the necessary competence to create and administrate all parts of the system.

### 1.6.2 Platform-as-a-Service (PaaS)

is the next step in the stack. It offers a development environment, consisting of programming languages and tools. Users can create their applications and they can deploy and run them directly and automatically in the cloud. The elasticity feature and the fault-tolerance are automatically offered by the system and they are managed by the provider. The developers can concentrate on the application development and management.

Paas increases the abstraction on the cloud and this is the main advantage of this solution. In fact, it allows higher-level programming with dramatically reduced complexity about the general/external aspects of the system. With this model users must only set the environment settings without thinking about the other low-level aspects. However, this can also represent a drawback, because the developer can only change these settings and therefore his possibilities are limited by the platform characteristics. In fact users do not control the physical infrastructure, the operating system and the low-level services.

As a consequence, this solution might for example not be appropriate for



situations where a particular property programming language, tool, software or environment are required. In addition the applications are closely linked with the platform and the services that this offers, so that a vendor lock-in problem must be considered. On the contrary, this solution is very convenient for multiple developers from all over the world that collaborate to create an application. In fact using a PaaS solution their work is shared immediately. Practically PaaS has two main shapes. The first is a Web interface that allows user to design, develop and deploy the application, the second is a programming APIs and a set of libraries that users must use during the development process. The most famous PaaS solutions is *Google AppEngine* that provide an API and various libraries and *Microsoft Windows Azure* that offer a platform to develop any type of applications on the top of the .NET technology.

Usually providers offers also a package to simulate the cloud environment on a local machine. The developers use this to test their applications during the development process. Once this phase is completed, they can easily deploy and run the applications on the cloud. In this way they do not consume and pay resources for incomplete applications.

### 1.6.3 Software-as-a-Service (SaaS)

is the top of the cloud stack. This acronym was used for the first time in 2001 by the *Software Information & Industry Association (SIIA)*. SaaS model include a series of applications ready to use. These are accessible through a web browser and an Internet connection. After the registration process, customers must only enter their credentials in the application website and they can immediately use them.

The applications are hosted in a cloud infrastructure that allows scalable feature and they are completely centrally managed by cloud providers. A SaaS version of most desktop applications can be already found on the cloud. A lot of different office automation, document management, social network and others SaaS are already widely used, often without users knowing that they

are using this type of cloud service.

In this delivery model users can only use the applications, without controlling any infrastructure parts or any operations during the development or after the application deployment.

SaaS is based on the Web 2.0 technology that allow to use a web browser as a normal application interface. SaaS is known as a *one-to-many* software delivery model, so the same application on the same infrastructure is used by multiple customers at the same time using the power of the cloud infrastructure.

SaaS is adapted for applications that for example are used from a huge amount of users. In fact a big advantage of SaaS is that it removes problems connected with software installation and software update. With traditional software each user must execute these operations on his machine, but with SaaS they are executed only on the cloud infrastructure by the provider transparently to the users. Another advantage is that SaaS applications are immediately accessible through any device that has a Internet connection and a web-browser without develop a specific version. In addition SaaS are advisable for users that need a software for a short period of time, in fact with SaaS users pay a subscription for the application access but not a usually considerable up-front costs for the licenses like for the desktop application. Clearly SaaS model is not the best solution for all applications, for example it is not a good fit when sensitive data should be inserted and stored or for software where a real-time response is required. These problems are the consequences of the execution on the cloud.

The most common example of SaaS is web-based email client. This solution is composed by a web interface accessible through a normal web browser that allows the user to execute the same operations of the desktop version. A large number of other applications can be identified as SaaS. Very much used in recent years are the cloud version of the office softwares. The most famous example is *Google Documents*. This service allows users to create, edit and manage a text document but also a presentation or other types of

file. Usually it is used with a cloud storage service like *Google Drive*. These services together give the user a complete solution in the office software field.

#### 1.6.4 Summary

As already mentioned, the development complexity depends on the model adopted. IaaS is the most complex. It is very flexible but the developer must manage all the aspects of the system and this requires a vast amount of knowledge to develop a good application. PaaS offers tools to develop and deploy the apps, so the developer can focus on the application logic, while all other aspects are managed by cloud experts. To conclude, SaaSs are ready applications accessible from the Web, the users need only to know the API of the specific application in order to use it.

### 1.7 Service-and-Compliance-level Agreements

The Service-and compliance-level agreements usually abbreviate in SLA is practically the contract between the provider and the consumer. It defines aspects of the service like the service quality or the user and provider responsibilities.

Usually the responsibilities vary depending on the delivery model adopted. For *SaaS* the user has less control so he is responsible just for what he does with the software. In the case of *PaaS*, the degrees of freedom for the user are higher, so consequently increasing also the responsibility. Finally in *IaaS* the user has total control of his system so he is responsible for all the operations executed with his machine.

### 1.8 Cloud Economic Aspects

The cloud economic aspects can be divided in two main sections. The first describes the economic advantages connected to the use of cloud computing,

the second instead analyzes the main pricing strategies adopted by biggest providers.

### 1.8.1 Economic Advantages

Cloud Computing has some economic advantages.

As mentioned above, cloud computing has an immediate economic advantage because it does not require any initial big investment to create an IT infrastructure.

But it has advantages also in the long period, in fact through cloud computing a company can shift the capital costs for the IT into operational costs that are more flexible and adaptable with the company situation. In particular there are some clear advantages, first of all thanks to cloud the company deletes the depreciation and the maintenance costs of the IT assets, it can reduce or delete the internal IT division with a big cost reduction and it does not have to pay software licenses but only a subscription to cloud services that is more flexible.

The company also removes all costs connected with the operativity of a private datacenter, like electricity or cooling.

If a company already owns some IT resources, it can use the cloud services only to manage workload peak or specific needs for a short time, without increasing its private infrastructure, which would require other big investments. In the future, if the company wants to move completely to the cloud infrastructure, it can then substitute its deprecated machine with cloud services.

### 1.8.2 Pricing Strategies

Cloud computing has three main pricing strategies.

In the first the customers pay depending on how many units of time they actually used the requested service. *Amazon EC2* adopts this strategy, with users selecting a machine type that has a specific cost per hour and paying

according to how many hours they used the machine.

In the second strategy each operation, like data transmission or memory allocation, has a cost and the user pay depending on the operations that he requests. In this case the developers should optimize their applications to limit the operations number to reduce the costs.

The last strategy is based on a subscription price. The customers pay a subscription and he can use the service for the amount of time established in the contract. This strategy is used mainly for the applications offered through the cloud (*SaaS*).

## 1.9 Reasons of Cloud Success

There are multiple reasons that have enabled the cloud computing success. They are linked to the advanced technologies available. In fact IT technologies are being constantly developed and in the last years software, networking, storage, and processor technologies have taken enormous steps forward. These allows cloud providers to offer better and more economic services, optimizing the usage of their infrastructures. Among the reasons for the success of the cloud we can find:

- the **financial advantages** for users thanks to the pay-per-use approach and the avoidance of buying and maintenance costs (operational costs related to IT software and infrastructure), as mentioned in 1.8.1.
- the **high performances** that users can obtain. Elaborations'time can also be shortened by requesting more resources and using the parallelization, i.e. relying on the cloud scalability in their applications.
- the **services availability**, as customers can access their data and they can use the cloud services everywhere, at any time and through different devices.
- the possibility to **react rapidly to unplanned IT resources needs**.

For these reasons cloud computing is considered by a vast amount of very different companies. Some of them already use the cloud for specific activities. For example, the *New York Times* used Amazon Cloud service, in particular *Amazon EC2*, to convert its digital library in a new format. This activity is perfect for cloud computing because it requires a considerable amount of computing resources for a short period of time, so it is very convenient to rent the necessary resources from the cloud. Thanks to the big amount of computing resources rented, the task required only 36 hours. After this period of time they do not incur in any other additional costs. Small enterprises and start-ups too, have a lot of reasons to use the cloud services. In fact they can obtain a complete IT infrastructure very quickly, without any big initial investment and without employing staff resources, that are usually limited in small companies, on the infrastructure. In the future, they can then easily expand their infrastructure, according to the business needs.

Other advantages can be found for the cloud providers. In fact they can take advantages of the infrastructure concentration into large datacenters. This allows cloud provider to save money optimizing the physical and staff resources.

## 1.10 Obstacles and major Challenges of Cloud Computing

In addition to the usual problems connected with the management of big IT systems, such as the configuration of all components, cloud computing presents other obstacles and challenges. These mainly depend on the dynamic resource provisioning feature and the usage of virtualization technologies, but also from other particular aspects of the cloud. Researchers are studying these problems to find solutions.

- One of the most common problems of using public cloud, is connected

with **confidential information security**. Confidential data in fact are stored and elaborated through an infrastructure managed by the cloud provider so they could be stolen from the providers themselves, as they are the ones that control the infrastructure, or from other users that share the same physical machine as a consequence of the multi-tenant technique when the security isolation is not perfectly implemented. Private clouds could be a solution for highly sensitive applications, but it is not always possible and as we already know they have some significant limits.

- Cloud users can freely request resources depending on their needs, but this can create a cloud overload with **availability problems and/or unpredictable performances**. This situation is inconvenient for all users and there are some types of applications that can not accept it. Usually providers ensure minimal performance parameters, described in the service-level agreements (SLAs).
- A big challenge connected with resource allocation is the development of a better and better **cloud resource management system** that optimize the usage of the cloud provider resources and offers users a better service in term of load balancing, QoS guarantees and capacity allocation.
- A problem connected with the huge amount of resources used in a cloud infrastructure and with the parallelization technique commonly employed in the cloud is **reliability**. In fact, cloud tasks can involve many nodes and - as it happens in a traditional system, but even more in a cloud system - node failures must be taken into consideration and handled. A better failure management gives benefits to providers, who can then optimize their resource usage, and to users, by accelerating tasks'executions.
- From the client's perspective, a performance problem is the **network speed**. In fact, in the last years data transfer bottlenecks connected

with low network capability, disadvantaged the cloud utilization. The network development and the creation of better network infrastructures in all advanced countries, are now reducing the impact of this problem.

- A big obstacle for a wider cloud spread is **the lack of standards**. This could be translated in a vendor lock-in problem, that involves very difficult and expensive operations to move an application from a provider to a new one.

The lack of standard also create problems when users try to use services from different vendors. As a consequence, the interoperability between services from different providers is today extremely difficult.

A standardization process between the bigger providers should resolve the problem and this will give users a bigger range of possibilities and a more flexible systems. On this theme some researchers imagine the cloud computing of the future as a totally open market, where the IT resources are traded and rented from the best bidder.

## 1.11 Cloud Vulnerabilities

All IT systems are target of malicious attacks. So the providers should protect their cloud infrastructures, that are nothing else than very big IT systems.

As the clouds spread, malicious individuals are developing new security threats and they are individuating new attack channels against cloud providers or cloud users. For example malicious users can try to take advantage of *multitenant access* supported by virtualization, that is a specific characteristic of cloud infrastructures.

The amount of resources available make cloud infrastructures a very interesting target for attackers that want to use these resources to execute large-scale attacks.

Some vulnerabilities examples are:

- Cloud is subject to **system availability threats**. An example is con-



nected with the *Internet domain name servers*. In fact the cloud services are available only through the Internet so a problem with these servers could make clouds inaccessible. In this case users can do nothing to resolve the drawback or to use the services, they must wait until the IDN servers return to work properly.

- However, cloud users - and in particular companies that use the cloud - must take care in the authentication and authorization of their employees handling the difference between their internal policy and the cloud policy. They must also try to prevent **account or service hijacking**.
- Cloud users should also consider and study the threats connected to **third-party data control** in a time where privacy is a well-felt topic. Sometimes, it is not easy to understand which type of situations users should consider, especially because it might not be clear who really controls the data (if there are subcontracts). For this reason the choice of storing important data on cloud could be risky.

Another problem that users usually do not think of, is connected with the deletion of confidential information stored on the cloud. In fact, users can not know if the data they intended to erase have been really deleted. In addition to this, users have no guarantee that even if the data are correctly deleted, the next user that will use the same hardware will not be able to recover some pieces of information.

Users do not have to only worry about external access to confidential data but also about problems connected with **data replication fails and storage media failure**. If important data are stored only in the cloud and a failure of this type happens, such data could be permanently lost.

On these themes the *Amazon Web Services customer agreement*, for example, does not help boost user confidence as it states: “We . . . will not be liable to you for any direct, indirect, incidental . . . damages . . . nor . . . be responsible for any compensation, reimbursement,

arising in connection with: (A) your inability to use the services . . . (B) the cost of procurement of substitute goods or services . . . or (D) any unauthorized access to, alteration of, or deletion, destruction, damage, loss or failure to store any of your content or other data”[6]. To ensure the secrecy of the data on the cloud someone think that *cryptography* can be a solution. Clearly it presents some advantages and some drawbacks as the system will require more time to process the information and this implies poorer performance. In addition it does not resolve completely the security problem. In fact encrypt data transmissions or encrypt data stored are safer than the non-encrypt versions but to process this information the cloud systems must decrypt the data and so these are easily accessible in the memory. In particular in systems based on the virtualization, extracting information from the memory is not too complicated.

- Cloud providers instead must protect their infrastructure from the most famous type of cyber-attack like **DoS and SQL injection**. An example of famous historic malicious attacks directed to cloud services is the one that occurred in May 2009, when a denial-of-service (DoS) attack was carried out against Google, affecting the availability of several of their services, including Google News and Gmail services, for various days [2]. Usually security professionals study the attacks and use digital forensics to identify and resolve the security flaws, but in a cloud environment where resources are shared between multiple users, this is much more difficult and sometimes impossible.
- An example connected with **physical problems** is “a lightning caused a prolonged downtime at Amazon on June 29 and 30, 2012” [2], but also any other catastrophic event or local problems in the area where a data-center is located could stop the correct operativity of it or even destroy the building. For these reasons and to have better performance everywhere, providers distribute the data centers in different geographical

areas and they use a massive data replication between them.

For further details on the cloud security theme, it is possible to visit the *Cloud Security Alliance (CSA)* website. “The Cloud Security Alliance (CSA) is the world’s leading organization dedicated to defining and raising awareness of best practices to help ensure a secure cloud computing environment” [8]. They periodically publish documents with the main cloud security threats with explanations, examples and suggestions.

## 1.12 Legal Issues of Cloud Computing

There are some problems connected with the cloud and the different countries’ laws. In fact in most cases the laws in force in many countries are old and may not be suitable to regulate this new environment.

Another problem connected with the international spread of cloud datacenters is that different countries usually have different laws in the data privacy field. This means that different situations are created depending on where the data are physically stored. European countries usually have stricter privacy laws while for example in the U.S. the governative agencies can have more freedom. On this theme the Russian Federation extended a law to oblige data operators to manage all personal data of citizens of the Russian Federation through systems that are in the territory of the Russian Federation. “*The Federal Law ‘On Personal Data’* Nr 152 dated 27 July 2006 (PDL), will be supplemented by a new requirement stating that data operators (entities performing functions of both controllers and processors, to use European terminology) are obliged (subject to certain exceptions) to ensure the recording, systemisation, accumulation, storage, clarification (update, change) and extraction of personal data of citizens of the Russian Federation with the use of databases located in the territory of the Russian Federation ” [9].

Probably an international regulations on matter and service-level agreements that provide more adequate legal protection for cloud users would be desirable.

## 1.13 Energy Use and Ecological Footprint

As mentioned above, the cloud infrastructure consists of a series of large data-centers. These consume a huge amount of energy and consequently they have an ecological impact.

According to Shehabi et al. (2016), “US data centers consumed about 70 billion kilowatt-hours of electricity in 2014, representing 2 percent of the country’s total energy consumption. That’s equivalent to the amount consumed by about 6.4 million average American homes that year” (For more details, the whole report can be accessed at the following link [7]).

These data are impressive, and they are growing steadily. It is estimated that in 2020 consumption will increase to 140 billion kilowatt-hours, costing about \$13 billion in power bills.

To reduce the electricity consumption, so to save money and protect the environment, a concept called *energy-proportional systems* is usually adopted. This method suggests to concentrate the workload in as few servers as possible. Thanks to this idea, the provider can put the unused machines in a low-power mode and use them only when necessary.

## 1.14 Cloud Applications

There are different application types that run on a cloud infrastructure. The most common are the web applications in general, but also processing pipeline tasks like image processing or batch applications like software automatic testing are executed in the cloud. In general all applications that require a big amount of resources for a small period of time could take advantage from cloud computing.

These applications cover a lot of different fields. Some examples are:

**Scientific field** usually applications in this field require very high performance. Cloud can satisfy this request with the possibility to add a potentially infinite amount of any types of IT resources. In addition

the cloud system is more flexible than a private system so the scientists can use for each experiment the most suitable infrastructure. For example the MapReduce programming model is often used to analyze the huge amount of data collected during experiments.

A clear example of the usefulness and convenience of cloud computing for science can be found in a *Microsoft Research* experiment, in which researchers took a biology experiment that in a normal infrastructure with 8 core CPUs, 14 GB RAM and a 2 TB local disk required about six CPU-years and tried to execute its on a cloud. They used more than 450 extra-large VMs from their data centers. The computation finished in 14 days thanks to the vast amount of resources allocated.

**Productivity field** many features traditionally offered by desktop applications are now available also through cloud services. These new solutions attract users because through cloud infrastructure the services but also the data are available anywhere, at any time, and from any Internet-connected device. In addition the cloud applications do not require any installation or maintenance operations.

Example of very popular cloud applications for document storage are *Dropbox* or *Google Drive*, while for the office automation the SaaS most used possibility is *Google Docs*. These Google services are hosted in the huge Google IT infrastructure and thanks to the cloud elasticity they have always the necessary resources to satisfy the customer's requests from all over the world.

**Social Networking field** in the last ten years social networks have been keeping records on the number of active users. This value has reached a huge amount and consequently a huge amount of IT resources are needed to manage all the user requests. Cloud computing is the solution adopted to solve this problem.



# Chapter 2

## Cloud Platforms

There are three main public cloud services vendors: Amazon, Google and Microsoft. These three possibilities will be analyzed in detail. During our experience, we directly used Amazon and Google services. Other proprietary or open source cloud solutions exist and are widely used, but they will not be treated in detail here.

### 2.1 Cloud computing at Amazon

With the spread of the cloud computing idea in the early 2000s Amazon began thinking of entering this market. The intuition behind its strategy was that it could use its powerful computing infrastructure not only for its business core, the e-commerce, but also to offer cloud service to increase the revenues. In 2006 Amazon was the first company that offered a public cloud service, in particular in the IaaS field with *EC2*, an elastic computing platform. In the following years Amazon extended its services and created *Amazon Web Services (AWS)*. AWS is a platform that offers a complete range of cloud services and today it is probably the most famous and the most used in the cloud world. It is growing steadily and in 2015 it became a \$10 billion business. Jeff Bezos, Amazon founder and CEO, described this platform: “AWS offers more than 70 services for compute, storage, databases,

analytics, mobile, Internet of Things, and enterprise applications. We offer 33 Availability Zones across 12 geographic regions worldwide, with another five regions and 11 Availability Zones in Canada, China, India, the US, and the UK to be available in the coming year”. This data display the dimensions of this platform and the fact that it is continuously developed and expanded. With its vast range of services AWS allows developers to easily create simple or complex applications with a lot of different programming solutions. To administer and monitor his account, his billing and the services requested the user can use the great Web-based console. This allows users to request any of the possible operations on the AWS world. It is possible to interact with the platform also through *SOAP* or *RESTful* Web service interfaces.

### 2.1.1 EC2 and Connected Services

*EC2* is the most famous possibility in the IaaS world. *EC2* and AWS have consequently become increasingly popular. As mentioned earlier *EC2* is an elastic computing platform that offers the possibilities to create and run virtual machine instances. Users can customize the virtual hardware requested or the software installed depending on their needs.

**Virtual Machine Characteristics** For the virtual machine hardware configuration users can choose one of the predefined standard virtual machine models or they can create a customized version where they select the amount of each type of resource, from CPU core to memory. There are a lot of different standard instances available with different characteristics for the most common needs. The models start from very basic machine to high performance versions. They are classified according to the main characteristics, with different types for *general purposes* or for *high-CPU* or *high-memory* or *high-storage* or *high-GPU* capability needs, and it is also possible select *micro instances* for applications with very low resource need. In each class the instances differ by the amount of resources available. For the pre-defined instances the prices start from \$0.0059 per hour and they can arrive to \$13.338



per hour but they depend on the zone selected. All these information can be easily found on the official website.

**Virtual Machine Basic Software** For the basic software installed when the machine is created, Amazon gives the possibility to choose between a vast range of precreated *Amazon Machine Images (AMI)*. These images provide the operating system, some libraries and some basic software. Users can select one existing *AMI* that is created by Amazon professionals so it is usually adapted for cloud and ready to use. But if they have particular needs they can also decide to create a personal *AMI* with customize software and using it in one or more of their instances to quickly recreate the same environment. On top of the *AMI*, users can install any software they need. It is important to remember that users have the root privileges so they can manage completely their instances and they can execute any type of operations like in a private machine.

**Virtual Machine Deployment and Management** When a user decides the resource characteristics and the *AMI*, he must specify the availability zones where he want to create the instance. This choice depends for example on the user preferences between minimizing costs or reducing communication latency or other characteristics. Clearly the instance cost per hour is tied with the amount of resources requested and it depends also on the zone where the instance is created. When the creation procedure is completed, the user can run and stop the instance in any moment through a command-line tools or the AWS console. To access the instance and to execute any type of operations with the root privileges in a safe manner the user can associate to the instance a key pair that allows him to control the system remotely. In addition, the user can control the access possibility to his machine through the firewall configuration and the security groups rules.

**Elasticity Service** We know that one of the most interesting cloud feature is the elasticity. To use this feature with the EC2 service, Amazon

offers an automatic scaling system called *AutoScaling*. This service regulate the instance characteristics or it creates new instances or delete the oldest depending on the workload without the user intervention. Users must only specify their personal conditions for example in terms of maximum costs or maximum or minimum number of instances activated simultaneously and the system handles the workload as well as possible respecting these indications.

### 2.1.2 Storage Solutions

Another important service that usually is used with EC2 to store and manage information is *Simple Storage System (S3)*. It provides distributed raw storage capability and it is based on buckets, virtual containers, that contain data represented through objects.

**Buckets and Objects** A bucket is stored in a specific datacenter inside an availability zone depending on the customer request. Usually it is replicated to avoid fault problems and for a better content distribution.

Objects can be anything from a simple file to entire disk images and they are stored in binary form.

Users can associate some metadata to the buckets or to the objects to characterise and to later easily individuate the information stored.

**Information Access** Information stored in *S3* are accessible from everywhere through a *Representational State Transfer (REST)* interface. So all the possible operations on the data can be translated in a HTTP request using the correct address to identify the target. Anything in *S3* is represented by a uniform resource identifiers (URIs) under the *s3.amazonaws.com* domain. So users can use GET or HEAD requests to obtain information, PUT or POST operations to insert information and DELETE to remove information from buckets, objects or metadata depending on the address used. The access is controlled through an *Access Control Policies (ACPs)*. Initially this allows only the owner to access the buckets or the objects, but he

can change the policies through the XML configuration file. The owner can grant the permission only to other *S3* users. To share the information with non-authenticated users, *S3* offers the possibility to generate signed URIs that allow requests for a limited amount of time from anyone that provide a correct temporary access token.

**S3 Limitations** *S3* service can be seen like a distributed file system so the information can be shared between different instances. It is necessary to note that *S3* has some limitations to obtain a highly efficient final result. The most significantly restriction is that the *S3* entity can not be manipulated. This service is thought for static information so renaming, modifying or relocating operations are not allowed. To execute any change users must remove the bucket or the object and he must completely re-upload the new version with drawbacks in terms of costs and time required.

Another inconvenience, although possibly less relevant, is that a *S3* solution is organized only in buckets that contain objects, it is not possible to create a file hierarchy through directory or subdirectory. This limitation can be easily solved using an adequate object name system. Each object must have a unique name in the bucket. This is the only characteristic required for the object names. It is important to note that path separators characters can also be used. So users can use the first part of the object names to represent what would be the object path in a normal filesystem.

**Amazon Elastic Block Store (EBS)** is a service that provide persistent storage to EC2 using the S3 service. The storage capability is provided through volumes that are mounted during the instances startup and that survive to the instances shutdown. *EBS* offers a different high-level S3 possibility usage.

**Database** In the database field Amazon offers different possibilities. Users can use their favourite third-party DBMS installing it with a correct *EC2 AMIs*, or they can use one of the available Amazon services in this field.

- There are many AMIs available that provide a DBMS. They cover all the most used possibilities like Microsoft SQL Server, MySQL, Oracle, PostgreSQL and others.
- The basic Amazon possibility is *Amazon SimpleDB*. This service offers a flexible relational database so it is suitable to store semistructured data. It is important to note that it has some constraints to obtain high performance also with big amount of data. It is based on three components:
  - **Domains** are the top-level element and they are similar to tables in the relational model.
  - The domains contain **items**, that are similar to relational model rows but with a flexible structure. Each item is a collection of attributes.
  - An **attribute** is a key-value pair.

So items in the same domain can have different attributes and this property makes this solution more flexible than traditional relational database.

- Another possibility is *Amazon DynamoDB*, a fully managed NoSQL database known for low latencies and scalability. It has a great documentation and it can easily interact with a lot of different programming languages.

### 2.1.3 Useful Services

Over the years the services offered are multiplying. Today a wide range of basic and sophisticated services are available and they allow users to create a complex and complete IT infrastructure. These include for example:

- networking support through *Amazon Virtual Private Cloud (VPC)*, *Elastic Load Balancing*, *Amazon Route 53* and *Amazon Direct Connect*

- caching systems through *Amazon ElastiCache*. It uses a Memcached-compatible protocol to give at EC2 instances a fast data access like a cache memory
- applications communications through *Amazon Simple Queue Service (SQS)* and *Amazon Simple Notification Service (SNS)*. These facilitate communications between applications deployed in the AWS infrastructure. *Amazon SQS* allows asynchronous communications using message queues and *Amazon SNS* is more specific for heterogeneous applications
- scalable and complete email service through *Amazon Simple E-mail Service (SES)*
- content delivery network through *Amazon CloudFront*. It is a service that offers an automatic way to implement a content delivery network for static or dynamic Web pages using strategically located Amazon virtual machines all over the world
- and others ...

A service that deserves a more specific analysis is *Elastic MapReduce (EMR)* because it is very useful and widely used. It allows developers to process large amount of data with the MapReduce programming paradigm. This offers an easy platform for MapReduce application supporting multiple programming languages. This service is based on *Apache Hadoop* as the MapReduce engine. The word “elastic” highlights the dynamism of this service where users can easily dynamically size the cluster or change the single machine characteristics according to the specific current needs.

All these services can be easily used together to create complex and flexible systems. Developers can use some specific services like *CloudWatch* to analyze, control, manage and optimize their applications and the services used. To find all the possibilities for each need, users can visit the official AWS website and the related documentation.

## 2.2 Cloud Computing: Google Perspective

In the cloud computing world Google is the most known vendor of Software-as-a-Service (SaaS). Probably its most used and famous application is *Gmail*. In 2016 Google announced that this service had more than 1 billion monthly active users. But Gmail is only one of the wide range of software that Google offers through SaaS delivery model. Other famous examples are *Google Drive*, *Google Docs*, and *Google Photo*. Some of the reasons why these applications are widely used are:

- usually they are free of charge for individual users,
- services are hosted in the Google datacenters that guarantee high performance and a fast access from anywhere in the world at any time, and
- Google reputation implicitly ensure the users about the great quality of the services.

For developers, SaaS applications are accompanied by a Platform-as-a-Service (PaaS) possibilities. The most famous service of this type of delivery model is *AppEngine*. Initially it supported only Python but today various programming languages can be used, like Java, PHP, NodeJS, Ruby and this list is constantly updated. This fact proves that Google is constantly investing and increasing its cloud offering.

This offering already contains also IaaS possibilities with a large range of various services for every need, although they are less famous and not widely used.

### 2.2.1 Google AppEngine

*Google AppEngine* is the main Google PaaS solution. So it offers a development and runtime environment to easily create scalable cloud applications. Clearly it offers convenient automatic solutions to use one of the main cloud

advantage, the elasticity, taking advantage of the large Google datacenters. But it gives also the possibility to use a collection of other services that can be integrated in an application to facilitate its implementation. So services like in-memory caching, messaging, or cron tasks can be directly used in the applications without implementing anything more.

**AppEngine Software Development Kit (SDK)** A very handy Google AppEngine feature is the possibility to develop the applications locally on the developer machine using the *AppEngine Software Development Kit (SDK)*. The main advantage of this possibility is certainly economic. In fact using the local version developers must not pay nothing during the application development and test. The *SDK* simulates perfectly the cloud environment so once the application is completed the developer need only upload the same code on the real cloud without changing anything and he can directly deploy the application. Google offers a terminal command that allows to migrate the local application on the cloud without the need for other operations. Developers can use their account settings to set costs limitations or performance constraints that the applications must respect.

**Infrastructure** The platform can be analyzed starting from the infrastructure. This is based on many servers within Google datacenters spread all over the world. Each HTTP request is sent to a server that host the specific application. This server handles the request and execute all the necessary operations. The infrastructure monitors the load of each application and if necessary allocates other resources to the existing machines or creates new machines to manage all the incoming requests.

**RunTime environment** The runtime environment is then deployed on the infrastructure. This is the layer where the applications are managed and executed. It has different responsibilities, one of them is certainly the isolation and the security of each application context and the servers security. This involves that each application is not influenced from others hosted in

the same machine and that the execution and the data stored are safe. In other words, it provides applications with a sandbox. However at the same time this component must also ensure the servers security from the threat coming from applications.

### 2.2.2 Storage Solutions

Applications that run on this environment have different storage possibilities. Each solution is specific for particular storage situation. Developers must choose the correct implementation depending on the volatility of the data and if these are static or dynamic. Other characteristics like required performance should also be considered. The three main classes are:

- in memory-cache
- storage for semistructured data
- long-term storage for static data

that usually are all used together.

In particular the most interesting situation is the storage of semistructured data. The Google service for this purpose is called *DataStore*. It manages data like objects, each of which is associated with a key used to retrieve the information stored. *DataStore* is similar to a relational database but the objects do not have a strict structure and this is the reason why this solution is suitable for semistructured data. Clearly this advantage has a drawback, as the absence of a predetermined schema involves some limitations on the available operations. These limits are nonetheless useful to ensure high performance also with large dataset and so they guarantee the scalable property.

### 2.2.3 Useful Services

The applications developed through AppEngine can use a vast range of already developed services to execute the most common operations without



writing new code. This implies that developers can use and assemble these services like small bricks to create the final applications.

A clear example can be found in the account management problem. In fact many different applications must implement this feature so Google offers the possibility to use the *Google Accounts Service*. It is developed and maintained by IT professionals, so it offers a great service. In addition it uses the Google's authentication system that guarantee high-level security standard. It is very flexible because it allows developers to associate profile settings like key-value pair to a Google Account. The key-value form allows the storage of any useful information for any application types. Developers can directly integrate this service in their applications without waste energy to reimplement a basic feature.

*Task Queues* and *Cron Jobs* are other two examples for asynchronous computation. The first allows a later execution of a task and the second gives the possibility to decide a specific moment when to execute the code that might not coincide with the instant when the web request is received.

But, as mentioned above, a vast range of other services is available.

#### 2.2.4 Develop, Deploy and Maintain process

Google supports developers during the complete application life cycle, from the development phases to the monitoring process after the application release, offering the necessary functionalities.

As mentioned above, developers can start building their applications using the *Google SDK* on their local machine. The *SDK* simulates perfectly the AppEngine runtime environment and it offers some functionalities to help the developers. Google offers different *SDK* depending on the programming language selected. For Java that is one of the most adopted language Google provides also a Google AppEngine plug-in for the *Eclipse* platform. Python, that is spreading in the last years, has a specific *SDK* that also contains an integrated Web application framework called *webapp*. It include particular features and tools that simplify and accelerate the application development.

Moreover it enforces a set of coherent practices to help non-expert programmers to realize well-coded applications. This does not mean that developers can not decide to adopt other frameworks to develop their applications.

After the development and testing phase has ended, each application can be deployed on AppEngine. Depending on whether the developer prefers to use the *web interface* or the *command-line tool*, there are different deployment procedures. Both ways are fast and easy. When the upload process on the Google Cloud ends, the application is immediately ready to use. AppEngine execute all the necessary operations through automatic processes, and the developer can directly use his application or manage it using the administrative console.

Among the possible operations, the developer can manage the application costs. For this purpose, the developer must know his application and the *Google Cost Policy*. In general, he can set a maximum daily budget that is composed by billable quotas, fixed quotas, and per-minute quotas. Developers can specify the values for each quotas but they can also divide the values for each resource to obtain a higher granularity. If a resource reaches the limit specified by the user, this will not be available to the application until the quota is replenished.

## 2.3 Microsoft Windows Azure

The Microsoft cloud offer is represented by Microsoft Windows Azure. It is a platform to develop and release applications in the cloud. As the two competitive services offered by Amazon and Google, it is a scalable runtime environment, particularly suitable for web applications but also for other types of software.

The physical infrastructure is spread in Microsoft datacenters all over the world. It is very convenient thanks to the basic facilities like compute, storage and networking services and the high-level facilities like for example access control or supply of business intelligence to the developers.

The main advantage is that all applications developed through Microsoft technology can be directly deployed in the cloud using Azure. So every application built on the Microsoft technology can be easily expanded through the scalability feature using the Azure platform.

The Azure world can be accessed and managed through the *Windows Azure Management Portal*. But developers who want to become familiar with the environment and to develop and test the applications, can simulate the Azure system on their own machines.

**The Azure architecture** includes a foundation layer that implements the structure and as mentioned above a set of developer services, from basic to high-level possibilities, that can be used to easily create complex applications by combining already implemented features. These useful facilities are collected in a middleware called *AppFabric*.

**AppFabric** has been described as “a comprehensive middleware for developing, deploying, and managing applications on the cloud or for integrating existing applications with cloud services” [1]. It offers an infrastructure that provides all the most useful cloud characteristics such as elasticity, high availability, sandboxing and multitenancy, and others. As mentioned above, *AppFabric* contains a set of services that simplify many of the common tasks in a distributed application. Examples are the access control through set of rules, or *Azure Cache*, a caching system to quickly access stored data.

### 2.3.1 Compute Services

Compute services are provided through three different options. Each of them has specific characteristics for a particular compute task type. The elasticity process is managed by Azure operating system. If there is a workload peak, Azure allocates new instances of the option selected and it distributes the incoming requests through a load balancer. The options are: *App Service*, *Cloud Services* and *Virtual Machines* and all information about them

can be found in the Microsoft Official Documentation [10].

- The Microsoft official documentation describes the **App Service** as “a fully managed compute platform that is optimized for hosting websites and web applications”. Developers can request the use of shared or dedicated virtual machines depending on their needs and their available budget. In both cases the isolation from other customers is always guaranteed. Different programming languages can be used to write the application code such as ASP.NET, the Microsoft solution for web programming, but also other possibilities such as Node.js, Java, PHP, or Python. An interesting feature could be the possibility to use PowerShell or other scripting languages. The *App Service* solution has some benefits in addition to the normal advantages of this type of services connected with the Microsoft world. For example, a *Visual Studio* integration is available to use the powerful Microsoft development environment to realize the applications, and direct connections to Microsoft SaaS platforms like *Office 365* are possible.
- The Microsoft official documentation describes the **Cloud Service** as a PaaS solution “designed to support applications that are scalable, reliable, and cheap to operate”. It is more free respect to *AppService*, so developers have more control over the software stack and they can install their own software. Clearly more control means less ease of use. It is important to remember that *Cloud Service* is a PaaS solution and not a IaaS solution with all the related consequences. This means that users do not request the virtual machines directly, but they create a configuration file that describes the necessary resources and the general application architecture and the platform will execute all the necessary operations to obtain the correct basic infrastructure. Users can obtain different infrastructure depending on their needs, simply changing the configuration file. The offer of *Cloud Service* is divided in two roles:
  - the *Web Role* that “automatically deploys and hosts your app

through IIS” and is therefore particularly suitable for Web applications,

- the *Worker Role* that “does not use IIS and runs your app standalone” and can be used for general compute services that do not use HTTP protocol to communicate through the Internet.

Simple applications are usually based on a single roles’instances but in complex applications both roles’instances can exist and work together. The applications created through this service are accessible through a single public IP address. The load balancer divides in a second moment the incoming requests between the virtual machines that are executing the target application. They are created depending on the workload, guaranteeing the elasticity feature in a way that always avoids a single point of hardware failure. *Cloud service* is fault tolerant, it handles physical server failure but also VMs and applications problems and it automatically executes all the necessary operations to resolve the situation.

- **Virtual Machines** is the Microsoft IaaS offering. It gives the possibility to create and completely control virtual machines. As all IaaS solutions, this service is useful to exploit the flexibility of virtualization without investing on physical hardware and on its maintenance. Users can decide on all the aspects of their machines, such as the location, the size, the operating system and others characteristics. Microsoft offers three different methods to create and manage virtual machines: a browser-based portal, a command-line tools with support for scripting, or directly using the APIs. It is important to remember that *Virtual Machines*, similarly to other IaaS solutions, is a low-level service, so users have complete control of VMs but this implies that they must manage the software stack through configuring, patching, installing and updating operations on all the software components.

### 2.3.2 Storage Solutions

All the computing services need storage services to maintain the data. Each instance of all computing possibilities has a local storage for temporary data useful for the current execution. But for durable and redundant storage, developers must use different services depending on their needs.

- The most general option is the usage of **BLOBs**. *BLOBs* or *Binary Large Objects* are adapted to store large text or binary files. Users can insert metadata to describe and recognize the information stored in the binary form. The possibility to create snapshot for backup purposes is very useful and convenient.
- For semistructured data Microsoft offers the possibility to create flexible **tables** without strict structures. The information are stored like rows in the table. Each row is a set of variable attributes and it is identified by a key that is the unique index for the table. This service is designed to manage large dataset and to give high-performance also for queries with huge result sets.
- A completely relational database based on the SQL server technology is available through **SQL Azure**. It has been adapted to work on the cloud providing scalable, highly available, and fault-tolerant features.
- The last service called **Queue Storage** allows application communications through permanent queues that maintain the content also between different applications executions. This feature ensures the developers against lost or unprocessed messages. Applications can write messages in the queue or read them through a FIFO (first-in, first-out) policy.

The data stored through all these services can be accessed simultaneously by different users from everywhere and they are geo-replicated to avoid data loss connected with major disasters.

### 2.3.3 Azure Peculiarity

*AppFabric* and the compute and storage services are the main components of the Azure platform. To conclude, a peculiarity of the Azure platform is the possibility to install it without differences on third-party data centers to recreate the Microsoft environment on a private infrastructure.

## 2.4 Other Possibilities

There is a long list of other cloud possibilities for any kind of need. For example:

**Apache CloudStack** “is open source software designed to deploy and manage large networks of virtual machines, as a highly available, highly scalable Infrastructure as a Service (IaaS) cloud computing platform. CloudStack is used by a number of service providers to offer public cloud services, and by many companies to provide private versions” [11]

**OpenStack** “is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface.” [12]

**Aneka** “is a platform and a framework for developing distributed applications on the Cloud. The infrastructure can be a public cloud available to anyone through the Internet, or a private cloud constituted by a set of nodes with restricted access.” [13]

**SalesForce.com** is a Software-as-a-Service solution based on the Force.com platform for customer relationship management (CMR)





## Chapter 3

# University Social Network

This chapter will briefly describe the design, the implementation and the deployment of a website developed during this project. In particular, we created a Social Network, taking inspiration from a *University of Pennsylvania* final laboratory project.

The main purpose of this Social Network was to collect data for the MapReduce elaborations and to show the results obtained. Clearly, the little time between the deployment of the website and the development of our MapReduce jobs meant that the amount of data collected was too small to execute “big-data analysis”. So as I will explain in the next chapter we realized one job on the SocialNetwork dataset but we also searched an external bigger dataset.

We chose to develop a SocialNetwork also because it is one of the most famous SaaS application type. As a consequence, in our project we worked with all cloud delivery models: IaaS to deploy the SocialNetwork, PaaS to develop the MapReduce jobs and SaaS that is the SocialNetwork itself.

The design, the implementation and the deployment required about three months of work, from February to April 2017.

## 3.1 Interface

The interface and the offered features were not a main part of this thesis project and for this reason they will not be explained in details. We developed a basic interface with the main useful SocialNetwork functionalities. As all website development, we used HTML, Javascript and CSS. In particular, we adopted *Bootstrap* and *Jquery* to simplify our work. External libraries were used to include some convenient and already implemented features. For example, we used:

- a library that offers an implementation of MD5 function for client-side encryption of the user's password, so to offer a more secure service
- and a library to manage the star rating of posts.

The back-end code has been developed in PHP using a library to communicate with the database.

The first webpage is a simple log-in page that allows users to access with an existing account or register a new one. Besides the usual pieces of information, the registration form requires the user to specify their University course. This is then used to automatically subscribe users to their course page. After the login, the user reaches the homepage showed in Figure 3.1.

The homepage offers a classic SocialNetwork interface. The main section of the homepage displays a form to create new posts, with the possibility to upload one or more photos or files, and a list of the most recent posts. Each post shows: the author, the data, the text content, the first image included, five empty stars that can be used to give a score to the post, a button to open the comments' section with the possibility to write a new one and two buttons to show the images and files uploaded, respectively. If the user is the author of the post, he can also remove the post using the "X" button on the top-right corner.

The section on the left of the page contains the user's photo, name and a list of buttons to navigate inside the SocialNetwork. These can be used to visit:

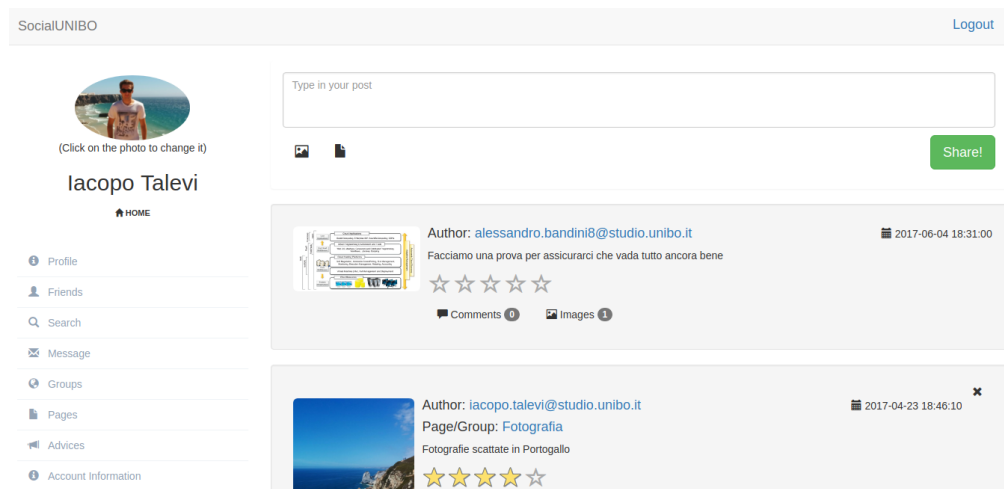


Figure 3.1: SocialUNIBO homepage

- the users' profile;
- the list of friend;
- an interface to search people, groups or pages, with the possibility to begin to follow them;
- a message page to read and write messages;
- an interface that shows the groups the user is following, and that gives the possibility to create new ones;
- an interface that shows the pages the user is following, and that gives the possibility to create new ones;
- a page with the account information;
- and the "Advice" page that is the most interesting for our project because it shows the MapReduce job results, as showed in Figure 3.2.

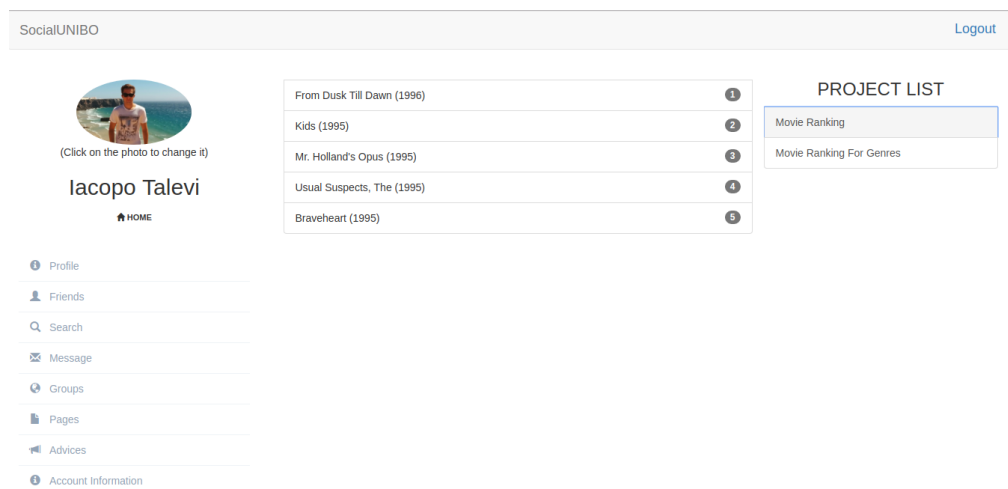


Figure 3.2: Advice Page

## 3.2 First Attempt using AWS

Initially, we tried to develop our system using Amazon Web Services. We spent some time investigating some interesting possibilities offered by this provider.

Amazon gives new users some free credits but also requires a credit card during the registration. As a consequence, if the user uses more than the available credits, their credit card is directly charged for any additional services, so students must always pay attention to the credit they have and not to consume more than their available budget.

We followed the indications from *University of Pennsylvania's* first laboratory class to create an Amazon Web Services account and to modify the settings in order to obtain the necessary environment to develop our project. For example, we had to create an access key to log in in our machines through SSH protocol.

### 3.2.1 EC2

With our account, we requested an EC2 instance to host our website. As mentioned in the Platforms chapter of this dissertation, EC2 is the first

and the most famous IaaS solution. So an EC2 instance is a virtual machine completely under user's control. I selected one of the available operating systems and installed the necessary packages.

Each EC2 instance has an IP address used to access the webpages hosted but also by developers to connect with their machines through SSH. We used the following command to access to the machine and to copy the updated code on it.

```
sudo ssh -i ~/.ec2/login.pem ec2-user@<IP address>
```

```
sudo scp -r -i ~/.ec2/login.pem ./* ec2-user@<PublicDNS>
```

We switch off the machine after each work session so not to consume our credits.

At this point we have the correct environment to begin the application development.

### 3.2.2 DataBase

A fundamental component of our application is the database. We used our database course project to design a first version of a relational database to manage SocialNetwork information.

Analyzing the Amazon services in the database field, we decided to use DynamoDB. “Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database, so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling” [14]. So we had to rethink and edit the initial database design, to pass from a relational version to a NoSQL solution, but the first project was still useful as a starting point.

Using the extensive documentation offered by Amazon official website and examples found on the Web, we created a first version of our database and

all the necessary queries.

An example of the code necessary to create a new table is the following:

```
var AWS = require("aws-sdk");
var user = {
  TableName : "User",
  KeySchema: [
    { AttributeName: "mail", KeyType: "HASH" } //Partition key
  ],
  AttributeDefinitions: [
    { AttributeName: "mail", AttributeType: "S" }
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 5,
    WriteCapacityUnits: 5
  }
};
var dynamodb = new AWS.DynamoDB();
dynamodb.createTable(user, function(err, data) { ... });
```

### 3.2.3 Amazon Elastic MapReduce

Once we had obtained a first very basic Social Network version, we decided to continue the development and at the same time began experimenting with data elaboration, in particular MapReduce, through the dedicated Amazon cloud services. In particular we selected Amazon Elastic MapReduce (EMR). “Amazon EMR is a managed cluster platform that simplifies running big data frameworks, such as Apache Hadoop and Apache Spark, on AWS to process and analyze vast amounts of data ”[15]. The excellent documentation offered by Amazon official website helped us to create our first example.

We tried to follow the first *University of Pennsylvania* laboratory task. This

requires to analyze the Wikipedia / DBpedia data set. This was our first MapReduce job and we debugged and tested it directly on the cloud, therefore spending our Amazon free credits. Testing directly in the cloud version rather than in the local version was unwise, as in this way we consumed all our credit to execute very basic tests.

### 3.2.4 The end of the Amazon Experience

At this point we were forced to abandon Amazon services because we had no credits left, and we therefore searched for other possible free solutions. In our short experience with Amazon services we noticed that they are very well documented. Together with the availability of many examples, this helps beginner developers to quickly get started with the offered services.

## 3.3 Application Development using our University Machines

While we were looking for a new solution, I asked technicians at our university for a virtual machine in our university servers that could simulate a IaaS cloud solution to continue the social network development. We obtained the virtual machine and I installed all the necessary packages as *Apache HTTP Server*, exactly as I would have done in a real IaaS solution. Although the interface we had already developed using HTML, CSS and JavaScript was compatible with the new infrastructure, the database had to be completely re-developed, as it was entirely based on Amazon services. This is an example of the “cloud vendor lock-in” problem, often cited among the main downsides of cloud computing at the present stage.

### 3.3.1 Cassandra

Eventually, we decided to use Apache Cassandra, a free and open-source distributed NoSQL database, designed to work with large amounts of data

across many commodity servers, while providing good performances and flexible solutions. In particular, the official website describes it as: “The Apache Cassandra database is the right choice when you need scalability and high availability without compromising performance. Linear scalability and proven fault-tolerance on commodity hardware or cloud infrastructure make it the perfect platform for mission-critical data” [16].

The Cassandra installation process and the library installation to manage the database through PHP was not easy, also because the documentation of the last version was incomplete.

We adopted the *DataStax PHP Driver for Apache Cassandra*, “a modern, feature-rich and highly tunable PHP client library for Apache Cassandra 2.1+ using exclusively Cassandra’s binary protocol and Cassandra Query Language v3” [17].

We had to adapt the database structure for the Cassandra characteristics and query limitations and as a consequence we ended up re-writing all the commands to generate the database and all the queries.

### 3.3.2 Test Phase

At the end of April 2017 we had a first official version deployed in our University datacenter, and asked friends and colleagues to test it, so that bugs could be identified and addressed.

## 3.4 Google Solution

While we continued the development using the virtual machine provided by our university, we found two potential alternatives. The first involved going back to use public cloud services, in particular Google platform, and the second consisted in obtaining access to a cluster of machines that would simulate a private cloud. We opted for the first solution, as it would give us the chance to experiment with another public platform and to use its specific services to execute the data elaboration section explained in the next chap-



ter.

We opened a project through a partnership between our University and *In-jenia* that gave us many free credits for Google Cloud Services. The social network code did not require any change, because in the second version it was based on a IaaS solution that is very flexible. This meant that we could continue the development on our university machine and not consume cloud credits.

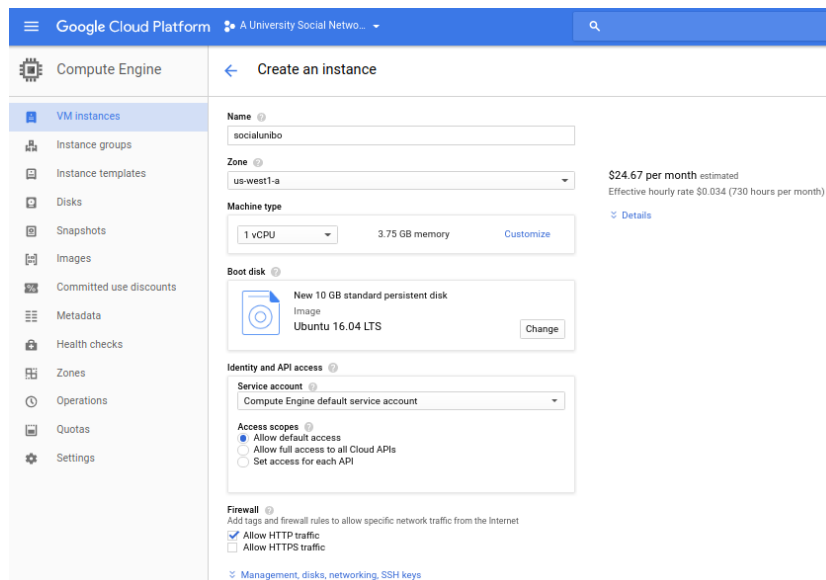
Only when the development was completed, I moved it into the cloud, re-creating the same environment.

### 3.4.1 Machines Characteristics

I requested a cheap machine configuration with 80GB of blank disk, deployed in the *us-west1-a* zone that had the best prices. As operating system I requested Ubuntu 16.04 LTS, because it is the same that our university technicians installed on our machine.

Clearly, I also requested that the HTTP traffic was allowed.

The request form is showed in Figure 3.3



The screenshot shows the 'Create an instance' form in the Google Cloud Platform console. The form is titled 'Create an instance' and is part of the 'Compute Engine' section. The left sidebar shows the navigation menu with 'VM instances' selected. The main form area contains the following fields and options:

- Name:** socialunibo
- Zone:** us-west1-a
- Machine type:** 1 vCPU, 3.75 GB memory
- Boot disk:** New 10 GB standard persistent disk, Image: Ubuntu 16.04 LTS
- Identity and API access:** Service account: Compute Engine default service account
- Access scopes:** Allow default access (selected), Allow full access to all Cloud APIs, Set access for each API
- Firewall:** Allow HTTP traffic (checked), Allow HTTPS traffic

The estimated cost is \$24.67 per month, with an effective hourly rate of \$0.034 (730 hours per month). The form also includes a 'Management, disks, networking, SSH keys' section at the bottom.

Figure 3.3: Compute Engine instance creation form

### 3.4.2 Packages Installation

When the virtual machine was created, I installed the necessary packages. I began with *apache2* and *php7.0*:

```
sudo apt-get install apache2 apache2-doc
sudo apt-get install php7.0 libapache2-mod-php7.0
```

to continue with Cassandra and the relative library.

At first the installation of Cassandra and the relative library gave some problems, but after different attempts I found the correct commands and the necessary file setting modifications. I stored them on a file so that I could easily re-create the correct environment to deploy our Social Network in any other virtual or real server.

```
echo "deb http://www.apache.org/dist/cassandra/debian 310x main"
    | sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list
curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add -
sudo apt-get update
sudo apt-key adv --keyserver pool.sks-keyservers.net
                    --recv-key A278B781FE4B2BDA
sudo apt-get update
sudo apt-get install cassandra
sudo service cassandra start

sudo apt-get install g++ make cmake libuv-dev libssl-dev
                    libgmp-dev openssl libpcre3-dev git php7.0-dev
git clone https://github.com/datastax/php-driver.git
cd php-driver
git submodule update --init
cd ext
sudo ./install.sh
sudo su
```

```
echo -e "; DataStax PHP Driver\nextension=cassandra.so" >>
    'php --ini | grep "Loaded Configuration" |
    sed -e "s|.*:\s*||"'
sudo nano /etc/php/7.0/apache2/php.ini
    Add in dynamic extension:
    extension=/usr/lib/php/20151012/cassandra.so

git clone https://github.com/datastax/php-driver.git
git reset --hard f50c93da3ea73ad8f8cf8b181d0313d437e559256
cd php-driver/ext
sudo ./install.sh
sudo reboot
sudo service cassandra restart
```

## 3.5 Conclusion

We tested the final version deployed on Google infrastructure, integrating it with the data elaboration results, and it worked properly. After a few test days we decided to delete the Google instance not to consume credits.

We tried to design and implement the Social Network with the idea of a real deployment in the future, taking into consideration that in that case it would have to work with larger amount of data, and would therefore need to scale easily depending on the needs. To this aim, we adopted solutions such as Cassandra, a powerful distributed database system, even though it was not necessary to manage the small number of test accounts we were dealing with. We applied this philosophy in every stage of our project.



# Chapter 4

## Data Elaboration

We live in the data-age so information is hugely important. For this reason, it is essential to have devices that can store as many data as possible and infrastructure that can quickly process them. It is very difficult to measure the exact amount of data stored today in electronic devices, but various studies foresee that in 2020 this value will reach 44 zettabytes that is  $44 * 10^{21}$  bytes [18]. To understand this amount, think that it is equivalent to  $\sim 5$  TB (i.e.  $5 * 10^{12}$  bytes) per person. Some scientific infrastructure like the *Large Hadron Collider (LHC)* at CERN produces, stores and manages about 30 petabytes (i.e.  $10^{15}$  bytes) of data per year [19].

One of the main problems of data storage and analysis is the storage devices access speed. Parallelization and distributed storage systems have been adopted to increase this value. These techniques solve the access speed problem but they create new situations that must be handled. For example the usage of many pieces of hardware introduce the hardware failure problem. This risk is usually addressed through replication techniques that create various copies of the data on different storage devices to avoid data loss. The *Hadoop Distributed Filesystem (HDFS)* is an example of storage system that implements these ideas.

The distributed storage means that during data elaborations, it could be necessary to combine data from different disks and this is usually not easy.

Different programming models, such as *MapReduce*, have been developed to automatically address this type of issues.

## 4.1 MapReduce

In the last years a new functional programming paradigm called *MapReduce*, developed by Google employees in 2004, has become widely used in the data elaboration field. This paradigm is particularly suitable for data-intensive batch applications that execute non-complex analysis on large data sets. It is not suitable for interactive analysis because its jobs could require minutes or more, for this reason it is mainly recommended for offline elaborations.

MapReduce was conceived for analyzing and processing a large amount of data in computer clusters taking advantage of *parallelization* in order to complete huge elaborations in a reasonable amount of time. In this way, it reduces the complexity of distributed systems management and coordination. For example, it relieves the developers from the burden of managing the parallelization, fault-tolerance, data distribution and load balancing.

One of the biggest advantage of MapReduce is that its applications can easily scale depending on the dataset size or on the cluster size to optimize performances.

As a consequence of the parallelization, this programming model is suitable for applications where the input can be easily partitioned into smaller blocks of similar dimensions that can be analyzed individually in parallel. The applications that entail this possibility are characterized by an input called *arbitrarily divisible input*. So it is important to highlights that MapReduce can not always be used in all situations or for all types of data elaboration. Sometimes the developers can adapt their data and their functions to take advantage of MapReduce, but in some circumstances this is impossible.

### 4.1.1 MapReduce Data Flow

MapReduce has two fundamental basic functions called *Map* and *Reduce*. They are written by users and executed in various copies on a cluster of machines in succession. Other library operations are also necessary during the elaborations. The MapReduce Data Flow can be summarized as follows:

- The input data are automatically partitioned and distributed between the Map invocations that execute the map function in parallel on each split. A Map invocation extracts useful information from each logical record contained in the corresponding input split using the user-defined map function and it creates a set of intermediate tuples (key/value pairs).
- Once all Map tasks are completed, the library merges all the tuples with the same intermediate key, producing an array for each key. These arrays are divided between the Reduce invocations through a partitioning function. For example “ $hash(key) \bmod N$ ” where N is the number of the Reduce instances requested by the user.
- A Reduce invocation elaborates in an iterative way the assigned arrays, that is in other words a key with a list of values, to create the desired output always in a key-value form. The user-defined reduce function is called for each array. Typically each Reduce invocation returns either one value or no values for each key. Each Reduce instance store its output on different file. So the result of a MapReduce job consists of multiple files. Users can choose to merge these files or to use them as inputs for another MapReduce job.

The *master* is a special node in the cluster that manages the job and assigns to others nodes, called *workers*, the tasks they must execute. It controls the state of each node to identify *idle workers* and assign them a map or a reduce task. It must also inform the instances about the location of input and intermediate files.

### 4.1.2 MapReduce Features

MapReduce is a *fault tolerant* programming model. The primary mechanism to manage these situations is the re-execution. This is possible thanks to the MapReduce model based on two separate functions. In particular the fault tolerant system is controlled by the *master node* that monitors all nodes in the cluster. It uses pings to periodically check that all nodes work correctly or to individuate possible failures.

The master node manages all Map and Reduce tasks through their states. The possible values are *idle*, *in-progress*, or *completed*. If a machine does not respond to a control ping and its task was in progress, the master detects a possible failure and it resets the task to idle. Also if the task is completed but the result was stored on the local disk, it resets that to idle. Thanks to these changes the task will be rescheduled as soon as an instance is available. The run-time environment, in addition to the handling of machine failures, automatically manages other aspects of parallel and distributed infrastructure. This allows programmers who are not expert in this field, to easily take advantage of data elaboration using large computer clusters. Examples of tasks automatically managed are: partitioning the input data, scheduling the job across the available machines and allowing communications between the machines in the cluster.

### 4.1.3 Examples

The most basic and elementary example of MapReduce is the problem of counting the number of occurrences of each word in multiple documents. A possible pseudo-code to resolve this problem can be taken from the official paper [20]:

```
map(String key, String value):  
    // key: document name  
    // value: document contents  
    for each word w in value:
```



```
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

A similar elaboration, but probably more useful in real life, is the problem of calculating the URLs access frequency analyzing the logs of a server. This can be easily achieved using the MapReduce programming model.

A possible solution is to analyze each row of the logs with the map function that extrapolates the URL and returns a pair with this value and the number 1 to report one access for that resource. All pairs with the same key, the URL in this case, are merged in an array and all arrays are passed to the reduce function instances. They only count the number of elements in each array, which corresponds to the number of times that the URL appeared in the logs. The value obtained for each array represents the number of times that the resource identified by the URL used as key has been visited, i.e. the desired result.

Another easy example that uses a maximum function instead of a sum for the reduce task is the calculation of the higher temperatures in various city starting from multiple files that contain a list of city names and relative measured temperatures. The job will use a map instance for each file that only takes each row and convert it in a key-value pair where the city name is used as key and the recorded temperature is stored as value. All the tuples with the same key from all files are merged together in an array. So at this point there are various arrays, each one associated with a single city. The reduce instances only search for the maximum values in each array and they return the name of the city linked with the array and the value found. The

list of these pairs is the desired result.

## 4.2 Hadoop

Hadoop is an open-source framework that provides a reliable, scalable platform for storage and analysis. It is hosted by the *Apache Software Foundation* and allows to process large data sets on commodity hardware. Hadoop is very convenient because it provides an automatic and complete environment. It allows to easily use the MapReduce model - developers must only specify the input data and define the map and reduce functions. The map and reduce input and output data are read or written through standard input and standard output. So any language that can use these two channels could be used to write a MapReduce application for Hadoop. To specify the key value pairs, it is sufficient to use a tab-delimited string.

Today, Yahoo controls probably the largest Hadoop cluster in the world with 40,000 servers that implement 4,500 nodes, followed by LinkedIn and Facebook. [21]

### 4.2.1 Comparison between Hadoop and other alternative systems

Hadoop is not the only implementation of a distributed system specific for data storage and data elaboration but it has some specific characteristics. It is possible to analyze them comparing Hadoop with other similar systems:

- *Distributed Relational Database* is convenient for dataset that are continually updated and where the queries require the analysis of a small portion of data. On the contrary Hadoop is convenient for data that are written once and read many times, and for analysis that use a big part of the dataset. In addition, it is more flexible because it is designed to analyze *unstructured data* such as plain text.

- *Grid computing* is another possibility. It is convenient for compute-intensive applications and it has some problems when large amount of data must be accessed. To address this type of situations, Hadoop implements a feature called *data locality*. The system distributes the jobs trying to assign to each node a job that uses the data stored on that node. So usually data can be accessed locally and this allows better performances. Moreover, grid computing programmers must explicitly handle the data flow and the failures using low-level routines, while Hadoop operates at higher level so the data flow and node fails is automatically managed.

### 4.2.2 Hadoop Distributed File System (HDFS)

Hadoop uses a distributed filesystem called *HDFS*, which stands for *Hadoop Distributed File System*.

This type of file systems manage the storage capability offered through a network of machines. *HDFS* is designed to work also in a heterogeneous networks composed by different hardware from multiple vendors. It is suitable to store very large files and to allows streaming data access patterns.

In particular it was conceived to work with the *write-once, read-many-times* pattern. The basic idea is that after a content is generated through a write operation, it will be used in different elaborations so many read operations will be requested.

In addition *HDFS* is adapted for analysis that involve a big part of the dataset, so it is optimized for this situation. As a consequence for example the latency in reading the first record becomes secondary.

### 4.2.3 Hadoop Data Flow

A Hadoop essential component is *YARN* (*Yet Another Resource Negotiator*). *YARN* is a cluster resource management system that implements an environment that allows the execution of any distributed program in a

Hadoop cluster.

A MapReduce job is composed by data that must be analyzed, the code that will be used to analyze the information and the configuration of the system. Hadoop divides the job in map and reduce tasks and they are scheduled through *YARN* and run on nodes.

As mentioned above, Hadoop manages all the work flow so it divides the input data in splits, creates a map instance for each split and so on. The dimension of the splits is an important parameter. In fact, if splits are too small the management time dominate the execution time, on the other hand if splits are too big the parallelization is not totally used. Usually a good split size coincides with the dimension of a *HDFS (Hadoop File System)* block because it is the largest size that can be guaranteed to be stored on a single node and the *data locality optimization* could be more easily obtained.

### 4.3 Analysis using Google App Engine

To execute MapReduce analysis on AppEngine a good solution is the *AppEngine MapReduce library*. “It is a community-maintained, open source library that is built on top of App Engine services, including Datastore and Task Queues” [24].

It provides an efficient system that can scale automatically. Developers only need to download and include the library in their applications and they can then focus on the application logic. Commonly needed operations such as partitioning the input data, scheduling execution across a set of machines and handling failures are automatically managed by the library.

The library is developed for Java and Python and it provides useful tools for developing and monitoring applications, as well.

This solution is similar in function to Hadoop but there are some differences between the implementations. These are listed by the official library wiki.

The users, as with any other AppEngine applications, will be charged for any resources that MapReduce programs realized through this library are going

to use.

### 4.3.1 Job Types

The library allows two types of jobs. In both cases, a job is composed by multiple stages and it uses the library already implemented features to efficiently and easily operate on large data sets.

The first type is the *Map Job*, which processes the data in parallel using only a single map stage.

More complex is the *MapReduce Job*. This has three stages performed in succession: map, shuffle and reduce. Each stage is executed only when the previous step is completed. This second type allows developers to completely perform a MapReduce data flow.

### 4.3.2 Stages

As mentioned above, there are different stages:

- The **Map stage** is controlled through a *Mapper class* offered by the library. It is composed by an *input reader* that extracts one logical record at a time from the input data and a *map function* specified by the user and called on each input record. The library offers different input readers depending on the input format but developers can build their own if needed. The Map function is more specific to the kind of job and it can create a final output if a Map Job is executed, or intermediate key-value pairs for a MapReduce Job.
- The **Shuffle stage** groups all the mapper output pairs that have the same key. All the merged values create an array that is associated with the common key. It does not delete multiple copies of a same value and it does not sort the values.
- These arrays arrive to the **Reduce stage** implemented by a *Reducer class*. The most important component is the *reduce function* called

for each key. It takes the key and the relative list of values as input and it elaborates them in an iterative way. The results of this function are passed to an *output writer* that generates the final result. As for readers, the library offers a range of possibilities for the writers as well, but custom version can be created for specific needs.

### 4.3.3 Input Reader and Output Writer

”The standard data input readers are designed to read in data from specific storage, such as blobstore or datastore and then supply the data to the mapper function. While the standard output writers write data from the reducer function to a Google Cloud Storage location” [25]. We used various Input Readers and Output Writers, in particular we tested:

**BlobstoreLineInputReader** requires only one parameter called *blob\_keys* that specifies a file stored in the Blobstore. It reads the specified file one line at a time and it passes the read value to the mapper function through a tuple comprised of the byte offset in the file and the line as a string. The format is: (byte\_offset, line\_value).

**BlobstoreZipInputReader** requires only one parameter called *blob\_keys* that specify a .zip object stored in the Blobstore. The .zip object contains multiple files, for each file the mapper function is called passing a tuple comprising information about the file and a function that allows the mapper to return the complete body of the file as a string.

**GoogleCloudStorageOutputWriter** requires, as other writers, a *bucket\_name* and if necessary other optional parameters as *content\_type*. It could create inconsistent outputs if the input changes during slice retries, but a different version that ensures this property is also available, if necessary.

In what follows, I will explain in some more details the BlobStore and the Google Cloud Storage mentioned above:

## BlobStore

“Google App Engine includes the Blobstore service, which allows applications to serve data objects limited only by the amount of data that can be uploaded or downloaded over a single HTTP connection” [26]. The stored objects are called *blobs* and they are created indirectly, by a submitted web form. The *Blobstore API* allows users and applications to access information in a file-like stream.

## Google Cloud Storage

Google Cloud Storage is one of the options for storing data in an App Engine application. It is suitable for files and their associated metadata. The storing is organized through *buckets*. “A bucket is the storage location you read files from and write files to. You must always specify a bucket when using the Google Cloud Storage client library” [27]. A project can access multiple buckets.

### 4.3.4 Configuration Settings

An important configuration parameter to use the parallelization is the *number of shards*. Shards are portions of input data that can be processed in parallel. A mapper instance is created to elaborate each shard. So the shard number specifies the number of mappers instantiated.

Using multiple configuration files, it is possible to completely customize all parameters. In our projects we modified two files to obtain the desired environment.

- The first was *app.yaml*, the main fields are:
  - a general section with the programming language used, python27 in our case, and the api version
  - a section for the requested resources where developers can specify the instance class and the scaling characteristics:

```
instance_class: B1
basic_scaling:
  max_instances: 8
  idle_timeout: 2m
```

in this case B1 is a machine with 128 MB of RAM and 600 MHz of CPU limit.

There are three different scaling techniques: we usually used the *basic scaling* where developers can specify the number of machines and a timeout, if a machine does not receive requests within the “idle\_timeout” this is shutdown. Other techniques are *manual scaling* and *auto scaling* that are well explained on the official documentation [28].

- a section to link URLs with the relative handlers
- a final section to specify the libraries and the files included
- The second was *queue.yaml* where developers can specify some settings of the queue task. The most important parameter is the *rate* that specifies how many tasks are launched each second.

All the configuration files are described in the official documentation that reports the possible fields and the possible values for each field [29].

## 4.4 MapReduce Pipeline example in Python

The basic class to perform a MapReduce elaboration through *Google AppEngine MapReduce Library* is the *MapReduce Pipeline* class that is used to “connect all the steps needed to perform a specific MapReduce job. It specifies the mapper, reducer, data input reader, output writer and so forth to be used to carry out the job” [30]. An example is:

```
class WordCountPipeline(base_handler.PipelineBase):
    def run(self, filekey, blobkey):
```



```
output = yield mapreduce_pipeline.MapreducePipeline(
    "word_count",
    "main.word_count_map",
    "main.word_count_reduce",
    "mapreduce.input_readers.BlobstoreZipInputReader",
    "mapreduce.output_writers.FileOutputWriter",
    mapper_params={
        "input_reader": {
            "blob_key": blobkey,
        },
    },
    reducer_params={
        "output_writer": {
            "mime_type": "text/plain",
            "output_sharding": "input",
            "filesystem": "blobstore",
        },
    },
    shards=16)
yield StoreOutput("WordCount", filekey, output)
```

The *MapReducePipeline* requires some parameters: the name of the job, the name of map and reduce functions, the input reader and the output writer with relative parameters and the number of shards (workers). The shuffle function is integrated and is not explicitly invoked.

To start a MapReduce job, it is sufficient to instantiate an object of the class defined and call the *start()* method.

Complete examples can be found in the official wiki. These provide a complete infrastructure that manages all the MapReduce necessary operations so they can be used as a basis for easily developing custom versions.

The wiki offers also a list of possible Input Readers and Output Writers, with short description, that developers can use in their applications.

## 4.5 Our Elaboration

**Installation** The first step was to install and configure the *Google Cloud SDK* and the necessary components to use Python and the MapReduce library.

*Google Cloud SDK* is “a set of tools to manage resources and applications hosted on Google Cloud Platform ”[31]. It allows to run local service emulators to simulate services for local development and tests and it allows to easily deploy the developed applications on the cloud. The official documentation provides the installation procedure, all the necessary materials and examples to use the *SDK*. To ensure us that all components work correctly we try the examples provided in the library wiki.

**Development** Starting from the examples code, which provides a basic infrastructure and a handy web-based interface that allows users to upload files in the *Blobstore* and to launch the analysis on them, we began to develop our applications. In particular, for each job we changed Map and Reduce functions and we adapted the pipeline definition to specify for example the input reader and the output writer that we wanted use.

**Dataset** We developed multiple analyses using data extracted from our database, as Social Ranking job, or using an external dataset. We searched datasets connected with the university world but we did not find any suitable possibilities. We found a large free film database with film information and user reviews that give us the possibility to execute different elaborations on the same data. It contains 27 thousand movies with 20 million ratings given by Netflix’s users.

**Testing** Each job has been developed and tested locally on our notebooks. Using the command: `dev_appserver.py ./app.yaml` the application is usable through a normal browser through the localhost address.

For the test phases usually we used a small portion of the dataset, the first

fifty rows, to execute the analysis in an acceptable time. In addition the local execution on a small portion of data allows a easier and faster debug process.

**Deployment** When we obtained a correct application version we deployed it on the cloud using the command line tool.

This requires to select a deploy zone/region. We encountered a problem because we had an "editor" role in the project created by *Injenia* and this role does not allow to select the deploy zone. The *Injenia* team resolved this problem making this choice automatic so particular permissions were no longer necessary. We asked to choose the cheapest region to save our credits and because we did not have particular performance needs. So each time the tool automatic selected the region and we only used the following command: `gcloud app deploy ./app.yaml` to deploy the application.

**Testing on the Cloud** On the cloud we used the entire dataset, loaded in .zip format using the offered interface, and the jobs required various hours. It is important to note that it was very difficult to check that the results of big-data analyses were correct, so we used the local execution on small portion of data to verify the solution correctness. Sometimes we found strange values in the final results connected with format errors in the input files, but the dimension of the input data does not allow to find all particular case to manage each of them.

The Social Ranking job dataset was already less than fifty rows so we analyzed the entire dataset locally and in the cloud to ensure that we could obtain the same results. This confirms that the adopted solution is usable also with large dataset using the cloud capacities.

### 4.5.1 Social Ranking

This job analyzes the data extracted from our database and in particular from the *follower* table to calculate a users rank. The main goal was to create a list of the most popular users using an algorithm based on the *PageRank*

*iterative algorithm.*

It takes as input a file generated by a query that contains a list of user mails pairs that represents a social network graph where vertices represent the users and the edges represent “follow” relationships. Each input line has the following format: <username, followedUser>.

### Formula

As mentioned above, we used an iterative algorithm where each round re-uses the values calculated in the previous round. We use the following formula:

$$r_i^{k+1} = d + (1 - d) \sum_{j \in B(i)} \frac{r_j^k}{|N(j)|}$$

where  $d$  is a constant parameter that we set equals to 0.15,  $B(i)$  is the set of users that follow  $i$ , the numerator is the rank of user  $j$  in the previous round and the denominator is the cardinality of the set of  $i$ 's followed users.

### Termination Criteria

The original algorithm terminates when the difference between the rank values in the current round and the rank values in the previous round for each user is less than a particular value. But we decided to also cap the number of iterations to a fixed number, to ensure that even in large dataset the task does not require too much time and/or resources. In particular we tried with values between one and ten iterations and we found five as a good compromise.

### MapReduce Tasks

In particular this job requires four different MapReduce tasks performed in succession. The first to prepare the input, the second to calculate the rank and it is repeated multiple times, the third to extract the useful information and to create a unique output file and the last to sort the result and translate

it in the JSON format to directly insert the desired rank in the social network interface.

The input file is read in the first task using the *BlobstoreLineInputReader* because the file is stored on the Blobstore. All other tasks use the *GoogleCloudStorageInputReader* and *GoogleCloudStorageOutputWriter* to read and write the intermediate values and the final result.

We found a problem to use the output of a task as the input of the following, because none of the examples provided in the library executed a chain of MapReduce tasks. We found a solution by adapting the name format of the output file to correctly use it in the second pipeline. In particular we realized two classes that translate a pipeline output into the correct format for the next one. We needed two different classes because if the first pipeline is only a Map task, an additional parameter is required to use the generated output. The code of these two classes can be found in the code section at the end of this dissertation.

The four tasks are as follows:

- The first task must prepare the input for the elaboration. So as first step using the Map function it converts each file line in a key-value pairs removing the hoop character. The Reduce function creates an element for each user with the correct format to apply, in the next step, the formula presented above. In particular it outputs a string: *username:rankValue—followedUsers*: where the initial rank value is 1 for all users and the followedUsers is a comma separated unique string with the list of users that follow the selected one.

```
"""Input: username<separator>followedUser
Output: <username,followedUser>"""
def social_ranking_count_map(data):
    (offset, line) = data
    #custom separator character
    separator = ","
```

```

if line != "":
    #last char is "\r" so we need to remove it
    yield (s.split(separator)[0],
           s.split(separator)[1][:-1])

```

```

"""Output: username:rankValue|followedUsers
followedUsers are comma separated in a unique string"""
def social_ranking_count_reduce(key, values):
    yield "%s:%s\n" % (key, "1|" + ','.join(values))

```

- The second task is repeated multiple times; it applies the formula to calculate the rank value of each user. The Map function and the Reduce function are not easily understandable. The Map function calculates the weight of the current user and yields this value to each user that he follows. It also yields the user and the list of followed just not to lose useful information and to pass them through different map-reduces jobs. The reduce function receives a user as key and a list of weights as values. These useful values are followed by the list of the followedUser that are required in the following iteration. It uses the try command to individuate and properly manage the weights and the strings. It sums the weights and applies the formula using the  $d$  parameter. This function prints for each user: *username:rankValue—followedUsers* exactly as in the previous task. In fact, this task is iterative so the output is used again as the input of the following iteration.

```

"""Input: user:rank|followedUsers
Output: <username,value> username is a followed user,
        value is the current user weight
Output: <username,followedUsers> useful for the next job"""
def social_ranking_map(data):
    (offset, line) = data
    if line != "":

```

```
tokens = line.split(":")
user = tokens[0]
info = tokens[1].split("|")
curRank = info[0]
followed = info[1].split(",")
#current user weight in ranking sum
count = float(curRank)/len(followed);
for f in followed:
    yield(f,float(count))
yield (user, followed)

"""Input: key = user
        values = [w1,w2,...wn,[followedUsers]]
Output: username:rankValue|followedUsers
        rankValue is the nth rank"""
def social_ranking_reduce(key, values):
    acc = 0
    user = ""
    for val in values:
        try:
            #if it is one of the weights, add it up
            sumPart = float(val)
            acc += sumPart
        except:
            #otherwise it's our list of followedUsers
            #remove initial '[' and final ']'
            users = val[1:-1]
            #create an array ['user1','user2',...]
            users = users.split(',')
            #remove initial whitespaces and quotes
            users = [u.strip()[1:-1] for u in users]
```

```

        #transform array into a string
        user = ', '.join(users)
    acc *= 0.15+(1-0.15)
    yield "%s:%s\n" % (key, str(acc)+"|" + user)

```

- The third task is only a Map task. It is executed with only one shard to generate the result on a single file. It takes as input the output of the previous task, which has the format required to apply the formula, and it extrapolates only the useful information: the user and the relative rank value. The code is in Appendix A.0.2.
- The last task is again only a Map task, that takes the list of users, each one with his rank value, and creates the sorted list. It returns this list in a JSON format. In this way, the output can be directly used in the Social Network to show the result. The code is in Appendix A.0.3.

### 4.5.2 Movie Ranking

The objective of this MapReduce job was to calculate the list of the best films depending on the available ratings. In particular it calculates the rating average for each film and with the obtained values it creates a sorted list with the first  $n$  films.

As in the previous job, the results are merged in a single file and converted in a JSON format to have the possibility to directly show the result on the Social Network.

#### Input

As mentioned above the film dataset contains 27 thousand movies with 20 million ratings given by Netflix's users. It consists of different files but we only used:

**movies.txt** where an example line is:

*1, Toy Story (1995), Adventure—Animation—Children—Comedy—Fantasy.*



It is composed by three comma-separated fields: <movieId,title,genres>, where the genres field is a pipe-separated string with the list of all film genres.

**ratings.txt** where an example line is:

*1,2,3.5,1112486027.*

It consists of four comma-separated fields: <userId, movieId, rating, timestamp>. Scores range from 0 to 5, and half scores are possible (0, 0.5, 1, etc).

### MapReduce Tasks

As the Social Ranking, this job requires multiple MapReduce tasks performed in succession. The first to calculate the average, the second to extract the desired information and to create a unique output file and the last to sort the result and translate it into the JSON format to directly insert the list of the best films in the social network interface.

The input file is read by the first task using the *BlobstoreLineInputReader*. All the other tasks use the *GoogleCloudStorageInputReader* and *GoogleCloudStorageOutputWriter* to read and write the intermediate values and the final result. As already explained for the previous job, we used two custom classes to use the output of a task as the input of the following.

**Problem** The first version of the job that used the *BlobstoreZipInputReader* required 7 hours with an average rate of 9918 mapper calls per second. We wanted to improve the performance of our job so we went through an accurate debug phase to check whether there were problems in our code. We noticed that although we specified in the pipeline to use 16 shards, only 2 shards are actually used during the execution, that is one for each file.

We did not find clear documentation about this issue so we were forced to analyze the code directly. We had this possibility because the library is open-source. We noted that the *BlobstoreZipInputReader* class contains code to automatically slice the input files into the requested shards (with a maximum

of 256 shards), but we also realized that this code had something wrong or it was never called. So the files were not sliced and only one shard for each of them was used.

Through the source code we found other Input Reader with small descriptions created through the comments. We decided to use `BlobstoreLineInputReader`. Thanks to this change the two files were sliced correctly in the requested number of shards. Using the same configuration files as before, so the same cluster characteristics the job ended in less than an hour, with an average rate of 30.000 mapper calls per second.

The three tasks used are the following:

- The first task performs a Join operation on the `movieID` column using the Map function and it calculates the rate average through the Reduce function. As a first step, it uses the Map function to return pairs where the key is always the `movieID` and the values can be the title or a score. The Reduce function use a try command to understand whether the analyzed value is the title or a score, and in the second case it uses this value to calculate the average.

```

"""Input: movieID<separator>title<separator>geners OR
    userID<separator>movieID<separator>rating<separator>timestamp
Output: <movieId,rating OR title>"""
def movies_ranking_map(data):
    (offset, s) = data
    #custom separator character
    separator = ","
    if s != "":
        if(s.find("\\" != -1):
            line_values = s.split('')
            #remove last char of id (it's a comma)
            movieID = float(line_values[0][:-1])
            title = line_values[1]

```

```
        yield (movieID, title)
    else:
        line_values = s.split(separator)
        try:
            #if the second field is a movieID,
            #save the rating.
            movieID = float(line_values[1])
            rating = line_values[2]
            yield (movieID, rating)
        except:
            #otherwise save the title:
            movieID = float(line_values[0])
            title = line_values[1]
            yield (movieID, title)

"""Output: title:averageRank"""
def movies_ranking_reduce(key, values):
    acc = 0
    count = 0
    title = ""
    for v in values:
        try:
            #if is a number, it is used in the average
            numericVal = float(v)
            acc = acc + numericVal
            count = count + 1
        except:
            #otherwise the value is the title
            title = v
    if(count>0):
        yield "%s:%.2f\n" % (title, acc/count)
```

```
else:
    yield "%s:%.2f\n" % (title, -1)
```

- The last two tasks are the same as explained for the Social Ranking job.

### 4.5.3 Movie Genres Ranking and Movie Suggestion

To conclude our project we decided to develop more complex elaborations on the film dataset that represents our “big data”. We developed two other MapReduce jobs that I briefly explain in this section, the code can be found in the code section at the end of this dissertation and a more complete explanation can be found in Alessandro Bandini’s dissertation.

#### Movie Genres Ranking

This job is an extension of the Movie Ranking job that considers also the film genres. In particular, it creates a film ranking for each genre used in the dataset.

The algorithm is based on the Movie Ranking version but the Mapper extracts more information from the film file, in particular it uses the movieID as key - as it did in the previous job - and the title concatenated with the genres list as value. The two components of the value field are separated by a custom character “#”, used in the second pipeline to split the different pieces of information. The Reducer calculates the average and it returns the title, the rate average and the genres list.

This output is passed to another Pipeline where the Mapper uses each genre (each film has usually more than one genre) as key and a pipe-separated pair composed by title and rate average as value. The reduce function receives for each genre a list of pairs <title—value> and it selects the films with the best overall score.

As mentioned for the first two jobs, we used two additional Pipelines at the

end to sort the result and to obtain a single file output that is written in JSON format.

### Movie Suggestion

This is the most sophisticated algorithm that we developed. It is based on “collaborative filtering”, used to perform an advanced elaboration to individuate the best film for a given user. In the real life, this is surely a more useful task for a user compared with a general global film ranking.

Collaborative filtering is a technique to make automatic prediction and choices about the interests of a person using preferences and taste information from many users that “collaborate” to take the decisions.

We divided this complex job in two smaller components:

- the first component identifies the best match between two users using the list of films voted with the same score. The idea is that the more scores they have in common, the more similar their film tastes are. The output is a pair for each user with his name and the name of the most similar user.
- the second component compares the list of films voted by the two users and record the differences between the two lists. Then it assigns to the first user those films that appear in the second user’s list but not in his own, i.e. the films that the second user saw but the first one has not seen yet.

Finding a better method to match similar users and consider more than one similar user can improve the results.

#### 4.5.4 Performance Analysis

We can not perform too many different tests with different configurations because we have a limited amount of credits. So we selected one job, *Movie Genres Ranking*, and we executed it with different situations to analyze the

performances. In particular we observed the time required by each phase. To save our credits we selected a cluster configuration that offers the necessary computing powers and that was not too expensive, and we always use that configuration in the tests. We then executed various experiments changing the number of shards.

Phases	Shard Number			
	8	16	32	64
Map Phase (min)	23	10	11	16
Shuffle Phase (min)	45	30	33	105
Reduce Phase (min)	0.58	1.11	1.58	3.38

Table 4.1: Execution time depending on the shard number

From the resulting data, it is possible to notice that the *Shuffle Phase* is the longest part independently from the number of shard requested. The *Reduce Phase* is the fastest and it requires a small part of the total time because it elaborates only the portion of data selected through the Mapper. In general the performances improve with the increment of the number of shards because the application exploit more the *parallelization*. It is interesting to note that this concept is only valid when moving from 8 to 16 shards, but the performances start decreasing already from 32 shards, and dramatically drop with 64 shards. This phenomenon is connected with the cluster dimension. In fact our cluster uses only 8 machines, so increasing the number of shards over a maximum limit increases the required work to manage the infrastructure and the parallelization level is counter-productive. The same observations can be done analyzing the average map calls per second (Table 4.2). A Map call is executed for each logic record in the input so more calls per second means a fast input elaboration.

A clearer picture of the required times can be obtained from the chart showed in Figure 4.1.

During the development in one of our first attempts, we used an Input Reader that does not slice the input and consequently it executes the job

8 Shards	16 Shards	32 Shards	64 Shards
14346	30483	29802	20270

Table 4.2: Average map calls per second depending on the shard number

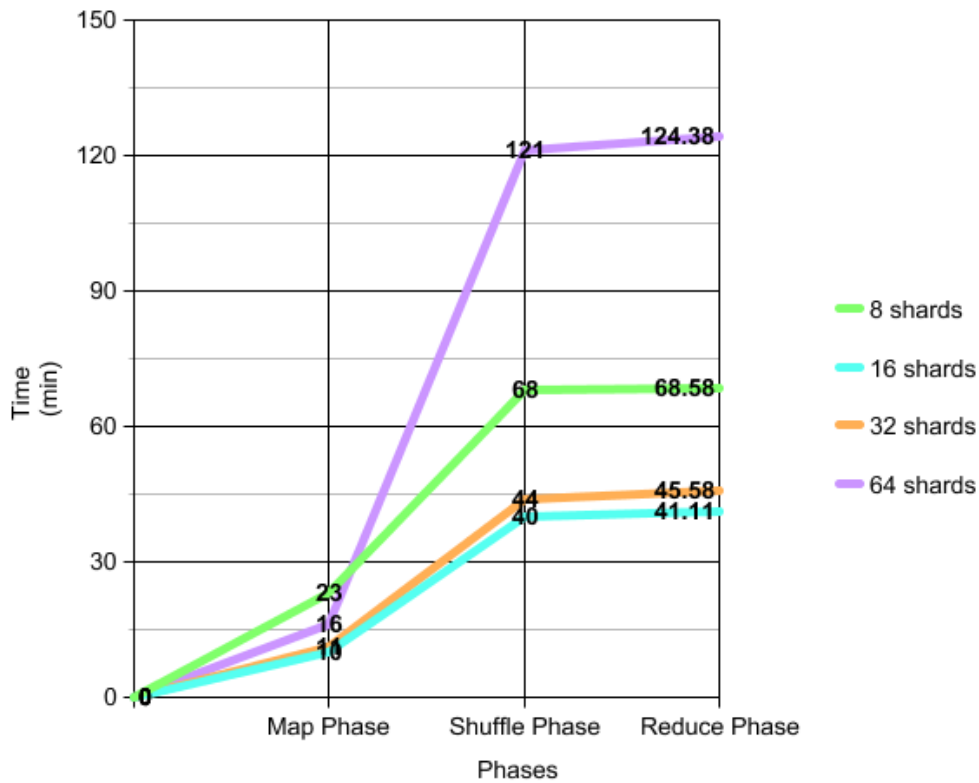


Figure 4.1: Total computation time with a different shard number

with only 1 shard. It requires about five hours and a half instead of the ten minutes required with 16 shards and it has an average number of map calls per second of 9918 against 30483 of the last version. These values show the power of parallelization even when not using huge dataset and when employing only limited resources.

## 4.6 Data Elaboration Conclusion

During the development and the testing of our applications we noted some positive and some negative aspects. The first positive aspect is that Google offers a sufficient amount of free credits to try its services. So students - but also normal users - have the possibility to experiment the services and to use them for small projects without paying. In addition, for the more common services, if a user requests less than a specific amount of resources these are free. A second positive aspect is that the Google Cloud Environment offers a large range of different automatic and already implemented services, very useful to solve every type of problems.

On the other side, we encountered some problems during the utilization of the MapReduce library. It is certainly a very useful library, thanks in particular to the just developed infrastructure and interface, but at the time this project was carried out the documentation was not up-to-date and the available examples were very simple and therefore scarcely informative for our purposes. This situation was partially solved thanks to the fact that the library is open source. So we could analyze directly the code that was well commented, and therefore solve our doubts.



# Conclusions

To conclude this dissertation, I would like to say that I am completely satisfied with the results of this project. I think that we have achieved all the goals we had set with Professor Zavattaro before and during the realization. We acquired theoretical knowledge of all aspects of cloud computing, from the more general aspects to the architectural and practical solutions. We learnt about the most common platforms, the differences between them and the services they offer. We directly used and explored different services from the most widely used public cloud platforms to realize a real usable application. We have also dealt with the big data analysis theme, analyzing and using a new programming paradigm. We realized, executed and tested multiple data elaborations through cloud computing.

I am also satisfied because I achieved the goals I had set myself not directly linked with the theme of the project. For example this experience has allowed me to confront myself with a long-term project, that required about one year of work in collaboration with a colleague. It has led me to face all stages of realizing a complex project: from the research and study of necessary materials, the design with technological, platforms and practical choices, the implementation with all the problems that are commonly encountered during this phase, to the testing of the final version. This general experience common to all projects will certainly be a useful building block for my future works.

In addition, during this work we committed some mistakes, chiefly due to the lack of experience, that forced us to re-design and re-implement parts of the

project. These operations required a lot of additional time. This convinced me even more of the importance of careful planning and the importance of realizing flexible codes that can be adapted to different situations.

During the realization of this dissertation I understood once more the importance of in-depth documentation and easily readable codes that make it easier to remember after a relative long period of time what solutions had been adopted and therefore make it easier to re-use or modify the realized parts.

The final application probably is not yet ready for a public large-scale deployment, also because at this stage it offers only basic features. Nonetheless, it requires only some small extensions to reach the goal mentioned above. It is also important to highlight that the Social Network actually works and we tested it with external users for a short-period of time and we did not find any relevant problems. We thought and designed also other features and tests that for example use different configurations, but we could not realize them due to a lack of time and limited available credits.

To conclude, I would like to highlight the topicality of this technology. I personally participated in a project at CERN (the European Organization for Nuclear Research) in June, July and August, where I discovered the current uses and development of various cloud solutions. After working on this university project, it was extremely interesting for me to witness how cloud computing is actually analyzed and adopted also in technologically advanced environments such as CERN.

# Appendix A

## Code

### A.0.1 Pipeline Adapter Classes

```
class GCSTMapperParams(base_handler.PipelineBase):
    def run(self, GCSPath):
        bucket_name = app_identity.get_default_gcs_bucket_name()
        return {
            "input_reader": {
                "bucket_name": bucket_name,
                "objects": [path.split('/', 2)[2] for path in GCSPath]
            }
        }
```

```
class GCSTMapperParamsAddBucketName(base_handler.PipelineBase):
    def run(self, GCSPath):
        bucket_name = app_identity.get_default_gcs_bucket_name()
        return {
            "input_reader": {
                "bucket_name": bucket_name,
                "objects": [path.split('/', 2)[2] for path in GCSPath]
            },
            "output_writer": {
```

```
        "bucket_name": bucket_name,
        "content_type": "text/plain"
    }
}
```

### A.0.2 Map Task to generate the result on a single file

```
"""Input: user:rank|followedUsers
Output: "username:rank" """
def social_ranking_final_map(data):
    lines = data.read()
    for line in split_into_sentences(lines):
        if line != "":
            tokens = line.split(":")
            user = tokens[0]
            info = tokens[1].split("|")
            curRank = info[0]
            yield "%s:%s\n" % (user, curRank)
```

### A.0.3 Map Task to sort the result and return it in JSON format

```
def getKey(item):
    if(item!= ""):
        return float(item.split(":")[1])

def sorting_map(data):
    global output_records
    lines = data.read()
    entries = split_into_sentences(lines)
    sortedEntries = sorted(entries, key=getKey,reverse = True)
    yield "{\n\"suggestion\":[\n"
```

```
count = len(sortedEntries)
if output_records > count:
    output_records = count
for i in range(0,output_records-1):
    yield "\"%s\",\\n" % (sortedEntries[i].split(":")[0])
yield "\"%s\\n" %
    (sortedEntries[output_records-1].split(":")[0])
yield "]\\n}"
```

#### A.0.4 Movie Genres Ranking

```
"""Input: movieID<separator>title<separator>genres OR
userID<separator>movieID<separator>rating<separator>timestamp
Output: movieID,rating OR title#genres"""
def movies_ranking_map(data):
    (offset, s) = data
    #custom separator character
    separator = ","
    if s != "":
        if(s.find("\\")!= -1):
            line_values = s.split('\\')
            #remove last char of id because it's a comma
            movieID = float(line_values[0][:-1])
            title = line_values[1]
            genres = line_values[2][1:]
            yield (movieID, title+ "#"+genres)
        else:
            line_values = s.split(separator)
            try:
                #if the second field is a movieID, save the rating
                movieID = float(line_values[1])
                rating = line_values[2]
```

```

        yield (movieID, rating)
    except:
        #otherwise save the title
        movieID = float(line_values[0])
        title = line_values[1]
        genres = line_values[2]
        yield (movieID, title + "#" + genres)

"""Output: title#genres:averageRank"""
def movies_ranking_reduce(key, values):
    acc = 0
    count = 0
    title = ""
    for v in values:
        try:
            #if is a number is used to calculate the average
            numericVal = float(v)
            acc = acc + numericVal
            count = count + 1
        except:
            #otherwise the value is the title
            title = v
            if(count>0):
                yield "%s#%.2f\n" % (title, acc/count)
            else:
                yield "%s#%.2f\n" % (title, -1)

"""Input: title#genres#averageRank
Output: <genre,title|rank>"""
def movies_genres_map(data):
    lines = data.read()

```

```
#custom separator character
separator = "#"
for s in split_into_sentences(lines):
    if s != "":
        line_values = s.split(separator)
        title = line_values[0]
        genres = line_values[1].split("|")
        rank = line_values[2]
        for g in genres:
            yield(g,title+"|"+rank)

"""Input: <genre,title|rank>
Output: genre:titles"""
def movies_genres_reduce(key, values):
    global output_records
    entries = {}
    for val in values:
        fields = val.split("|")
        entries[fields[0]] = fields[1]
    sortedEntries = sorted(entries.items(),
                            key=operator.itemgetter(1),reverse=True)
    titles = ""
    count = len(sortedEntries)
    for i in range(0,output_records):
        if i >= count:
            break
        else:
            titles += ",\\" +
                (''.join(str(sortedEntries[i]).split(",")[:-1]))[2:-1]
            + "\\"
    titles = "[" + titles[1:] + "]"
```

```

        yield "\"%s\":"%s\n" % (key,titles)

"""Input: genres:[filmList]
Output: the same string"""
def movies_genres_final_map(data):
    lines = data.read()
    for line in split_into_sentences(lines):
        if line != "":
            yield "%s\n" % (line)

def sorting_map(data):
    lines = data.read()
    yield "{\n\"movieGenresSuggestion\":{\n"
    entries = split_into_sentences(lines)
    count = len(entries)
    for i in range(0,count-2):
        yield "%s,\n" % (entries[i])
    yield "%s\n" % (entries[count-2])
    yield "}\n}"

```

## A.0.5 Movie Suggestion

### Find similar users

```

"""Input:
userID<separator>movieID<separator>rating<separator>timestamp
Output: <movieId, usrId-rate>"""
def rated_movie_map(data):
    (offset, s) = data
    #custom separator character
    separator = ","
    if s != "":

```



```
        line_values = s.split(separator)
        userId = line_values[0]
        movieID = line_values[1]
        rating = line_values[2]
        yield (movieID, userId + "-" + rating)

"""Output: movieID:[usri-.-.-usrk] [usrx-.-.-usry] """
def rated_movie_reduce(key, values):
    rateDict = {}
    for v in values:
        entry = v.split("-")
        if(entry[1] in rateDict):
            rateDict[entry[1]] = rateDict[entry[1]] + "-" + entry[0]
        else:
            rateDict[entry[1]] = entry[0]
    result = ""
    for rate in rateDict:
        result = result + "[" + rateDict[rate] + "]"
    yield "%s,%s\n" % (key, result)

"""Input: movieIDseparator>[u1-u2-.-.] [users]..[users]
Output: <usr1, usr2>"""
def affiliated_user_map(data):
    text = data.read()
    #custom separator character
    separator = ","
    for s in split_into_sentences(text):
        if s != "":
            values = s.split(separator)[1]
            affiliated = values.split("[")
            for aff in affiliated:
```

```

        #remove last char ("]")
        aff = aff[:-1]
        users = aff.split("-")
        count = len(users)
        for i in range(0,count):
            for y in range(0,count):
                if i != y:
                    yield (users[i],users[y])

"""Output: user1:user2
       where user2 it's the best match for user1"""
def affiliated_user_reduce(key, values):
    affDict = {}
    for v in values:
        if(v in affDict):
            affDict[v] = affDict[v] + 1
        else:
            affDict[v] = 1
    max = 0
    bestUser = key
    for user in affDict:
        if affDict[user] > max:
            bestUser = user
    yield "%s:%s\n" % (key, bestUser)

"""Input: title:averageRank
Output: the same string"""
def final_map(data):
    lines = data.read()
    for line in split_into_sentences(lines):
        if line != "":

```

```
yield "%s\n" % (line)
```

## Movie Filter

```
"""Input: <userID1,idFilms> OR
userID<separator>movieID<separator>rating<separator>timestamp
Output: userWhoSuggest, U:userToSuggest OR M:movieToSuggest"""
def movies_suggestion_map(data):
    (offset, s) = data
    #custom separator character
    separator = ","
    if s != "":
        values = s.split(":")
        if len(values) > 1:
            #we are in the <userID1,idFilms> file
            usr = values[0]
            films = values[1].split(",")
            for f in films:
                yield (usr,f)
    else:
        values = s.split(separator)
        usr = values[0]
        movieID = values[1]
        yield (usr,movieID)

"""Output: usr:suggestedFilm"""
def movies_suggestion_reduce(key, values):
    movies = {}
    for v in values:
        if(v != ""):
            if(v in movies):
```

```

        movies[v] = movies[v] + 1
    else:
        movies[v] = 1
result = ""
for m in movies:
    if movies[m] == 1:
        result = result + "," + m
yield "%s:%s\n" % (key, result)

```

"""Input: title:averageRank

Output: the same string"""

```

def final_map(data):
    lines = data.read()
    for line in split_into_sentences(lines):
        if line != "":
            yield "%s\n" % (line)

```

## Suggestions

"""Input: <userID1:userID2> OR

userID<separator>movieID<separator>rating<separator>timestamp

Output: userWhoSuggest, U:userToSuggest OR M:movieToSuggest"""

```

def movies_suggestion_map(data):
    (offset, s) = data
    #custom separator character
    separator = ","
    if s != "":
        values = s.split(":")
        if len(values) > 1:
            #we are in the <userID1:userID2> file
            usr1 = values[0]

```

```
        usr2 = values[1]
        yield (usr2,"U:"+usr1)
    else:
        values = s.split(separator)
        usr = values[0]
        movieID = values[1]
        yield (usr,"M:"+movieID)

"""Output: usr:suggestedFilm"""
def movies_suggestion_reduce(key, values):
    users = []
    movies = []
    for v in values:
        if(v != ""):
            if v[0] == 'M':
                movies.append(v.split(":")[1])
            else:
                users.append(v.split(":")[1])
    for u in users:
        for m in movies:
            yield "%s,%s\n" % (u, m)

"""Input: usr<separator>suggestedFilm
Output: <usr1, film>"""
def filter_suggestion_map(data):
    text = data.read()
    #custom separator character
    separator = ","
    for s in split_into_sentences(text):
        if s != "":
            values = s.split(separator)
```

```
        yield (values[0],values[1])

"""Output: user:filmSuggested (comma-separated)"""
def filter_suggestion_reduce(key, values):
    yield "%s:%s\n" % (key, ",".join(values))

"""Input: title:averageRank
Output: the same string"""
def final_map(data):
    lines = data.read()
    for line in split_into_sentences(lines):
        if line != "":
            yield "%s\n" % (line)
```

# Bibliography

- [1] Mastering Cloud Computing Foundations and Applications Programming (Rajkumar Buyya, Christian Vecchiola and S. Thamarai Selvi)
- [2] Cloud Computing: Theory and Practice (Dan C. Marinescu)
- [3] Hadoop: The Definitive Guide, 4th Edition Storage and Analysis at Internet Scale (Tom White)
- [4] MapReduce Design Patterns Building Effective Algorithms and Analytics for Hadoop and Other Systems (Donald Miner, Adam Shook)
- [5] <http://www.cis.upenn.edu/nets212/>
- [6] <https://aws.amazon.com/it/agreement/>
- [7] <https://eta.lbl.gov/publications/united-states-data-center-energy>
- [8] <https://cloudsecurityalliance.org/about/>
- [9] [https://united-kingdom.taylorwessing.com/globaldatahub/article\\_dp\\_cyber\\_russia.html](https://united-kingdom.taylorwessing.com/globaldatahub/article_dp_cyber_russia.html)
- [10] <https://docs.microsoft.com/en-us/azure/cloud-services/cloud-services-choose-me>
- [11] <https://cloudstack.apache.org/>
- [12] <https://www.openstack.org/software/>
- [13] [http://www.manjrasoft.com/aneka\\_architecture.html](http://www.manjrasoft.com/aneka_architecture.html)

- 
- [14] <http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>
  - [15] <http://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-what-is-emr.html>
  - [16] <http://cassandra.apache.org/>
  - [17] <https://github.com/datastax/php-driver/>
  - [18] <http://www.northeastern.edu/levelblog/2016/05/13/how-much-data-produced-every-day/>
  - [19] <https://home.cern/about/computing>
  - [20] <https://research.google.com/archive/mapreduce.html>
  - [21] <https://wiki.apache.org/hadoop/PoweredBy>
  - [22] <https://www.slideshare.net/chopramanish/organizations-with-largest-hadoop-clusters>
  - [23] <https://github.com/GoogleCloudPlatform/appengine-mapreduce/wiki>
  - [24] <https://cloud.google.com/appengine/docs/standard/python/dataprocessing/>
  - [25] <https://github.com/GoogleCloudPlatform/appengine-mapreduce/wiki/3.4-Readers-and-Writers>
  - [26] <https://cloud.google.com/appengine/docs/standard/python/blobstore/>
  - [27] <https://cloud.google.com/appengine/docs/standard/python/googlecloudstorageclient/understanding-storage-features>
  - [28] <https://cloud.google.com/appengine/docs/standard/python/an-overview-of-app-engine>
  - [29] <https://cloud.google.com/appengine/docs/standard/python/configuration-files>



- [30] <https://github.com/GoogleCloudPlatform/appengine-mapreduce/wiki/3.3-The-MapreducePipeline-Class>
- [31] <https://cloud.google.com/sdk/docs/#more-information>



# Acknowledgements

I would like to express my gratitude to Professor Zavattaro for the interesting possibility that he offered me, for his advices and continuous support during the realization of this project.

I am also indebted to Alessandro Bandini, who collaborated with me on this project and who often sacrificed even his free time to work with me on it.

Finally, my heart-felt thanks to my sister for her advices during the writing of this dissertation and for always believing in me, and to my family and friends in general, for their constant support and encouragement throughout my university path away from home.