2 Background

To clearly understand how cloud computing technologies are combined to collaborative filtering methods to achieve the main contributions of this thesis, tackling some limitations of traditional recommendation architectures, it is important to have first an overview of the fundamentals of recommender systems and cloud computing. This chapter presents essential background information regarding recommendations using collaborative filtering, and also introduces key cloud computing concepts and platforms used in the development of this work.

2.1. Recommender Systems

Recommender systems are used in wide different web applications, mostly in order to predict user preferences, being by products, contents, places, etc. Predictions with good accuracy may result in increased revenues and also could improve user satisfaction and fidelity to services. The main functions of recommender systems are to analyze user data and extract useful information for future predictions [12]. There are several techniques for implementation of recommender systems, including content-based, collaborative filtering, among others. To improve the performance of predictions, these methods can be combined in hybrid systems. This section will explore those methods.

There are several systems accessible via the Internet aimed at generating recommendations for various types. For example, recommender systems are now a key part of several e-commerce portals like Amazon.com [3] and iTunes Store [24], and also are an important component in services like Netflix [4], Pandora [25], Spotify [26], etc. In general, these systems seek to acquire opinions or preferences about items in a group of users, and use these data to display items that might make sense for other users.

From this overview one can see that the recommender systems need basically two things to function properly:

1. Information about the preferences of users
2. A method for determining whether an item is interesting to an user

Typically, user preferences include external information such as personal characteristics (age, sex, location, etc.), its history of interactions with the portal in question, and their ratings on the items [13]. The way to determine if an item is interesting for a user or not depends on the type of recommendation system. In general, all recommender systems follow a well-defined process to create recommendations.

Considering the recommendation process as a black box, as shown in Figure 1, one can identify two sources of information necessary to process input. These information sources correspond to user data and information about the items. Ideally, these data related to the user profiles should be explicitly provided by the users. However, this information can also be implicitly extracted from other sources such as browsing through the pages, consumption or purchase items, how items were consumed, if a user watched or not a movie, how long the movie was watched, etc.

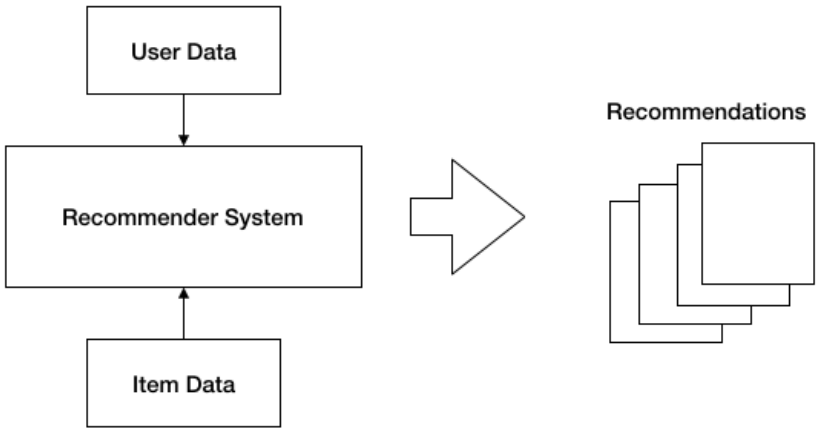

Figure 1 - Recommendation Process

2.1.1. The Recommendation Problem

Recommendation methods have been informally used for years. Positive and negative recommendations help people discover new things they would like or avoid bad alternatives that they would not like.

Recommender systems have emerged as an area of independent research in the mid 1990s, when researchers began to focus on recommendation problems that explicitly relied on evaluations [27, 28, 29]. Over the past decade, much has been done both in industry and academia in developing new approaches to recommender systems and interest in this area still remains.

Informally, the recommendation problem could be reduced to the problem of estimating ratings for the items that have not been seen/evaluated by a user. This rating estimation is usually based on the ratings already given by the user to other items and on some other information that will be formally described later. Once computed estimated ratings for the yet unrated items, it is possible to recommend to the user the items with the highest estimated ratings.

Formally, let C be the set of all users and let S be the set of all possible items that can be recommended, such as products, books, movies, or web pages. Normally both sets are very large, and, as discussed in following of this thesis, could contain tens of millions of objects. Let u be a utility function that measures the usefulness of item s to user c, i.e., $u : C \times S \rightarrow R$. Here, R is an ordered set, and may contain non-negative integers, or real numbers in a certain range, which depends on the representation scheme employed in the recommendation system [8].

Using the model described, the recommendation problem is reduced to choose an item $s' \in S$ for an user $c' \in C$ who maximizes the utility function:

$$\forall c \in C, \quad s'_c = \arg\max_{s \in S} u(c, s).$$

Each user in space C can be defined using a profile. This is the so-called user profile, containing information about the user's tastes to the recommendation domain. The user profile representation depends on the approach and the field, but for example, it can include features such as age, gender, income, marital status, among others. In a most simple case, the user profile can contain only a single element, such as the user ID.

Similarly, each item in the space S can be defined as a set of characteristics, depending on the chosen representation scheme. For example, in a video recommendation system, S can be a collection of movies. Each of the movies in S

can be represented not only by its unique identifier, but also for its content information, such as its title, genre, director, year of release, main actors, among others, which is thought to have an effect on the recommendation process.

The central problem of recommender systems lies in that utility u is usually not defined on the whole $C \times S$ space, but only on some subset of it. This means u needs to be extrapolated to the whole space $C \times S$. The recommendation engine should be able to estimate the ratings of the non rated item-user combinations and issue appropriate recommendations based on these predictions.

For example, in a movie recommendation system, users initially evaluate some subset of movies they have already seen. An example of the user evaluation matrix for the items for a movie recommendation application is shown in Table 1, where evaluations are specified as 1, positive, or -1 negative. The *(-)* means that users did not evaluate the corresponding movies.

	Movie A	**Movie B**	**Movie C**	**Movie D**
User A	1	1	1	-
User B	-	-	-1	-
User C	-1	-	-	1
User D	-	-1	1	-

Table 1 - Rating Matrix for movies

Therefore, the recommendation engine should be able to estimate or predict the evaluations of these user-movie combinations and then make appropriate recommendations based on these predictions. Considering the space $C \times S$ as an array of assessments, many assessments are not initially provided, so the matrix is said to be sparse. Thus, the work of recommendation system is extrapolating u to the entire space $C \times S$. The process to make these extrapolations defines the utility function. This extrapolation task has been addressed in various ways [7, 29, 30]:

1. Specifying heuristics that will define the utility function empirically by validating their performance.

2. Estimating the utility function by optimizing pre-defined performance criteria, such as the root mean square error [8].

2.1.2. Recommendation Strategies

The recommendation techniques have a number of possible classifications [1, 7, 31]. Specifically, the recommender systems have:

- context data, the information that the system has before the recommendation process starts;
- Input data, the information that users need to communicate to the system in order to generate recommendations
- an algorithm which combines information on the context and the input data to produce suggestions.

Thus, the most common recommendation techniques may be grouped into two distinct categories. Content filtering approach seeks to create a profile for each user or item in order to characterize their nature. For example, in case the item is a video, one can use the channel where it is displayed, the program to which it belongs, the actors participating and other data to the formation of this profile to then recommend other videos fit within this profile. The same goes for the creation of user profiles, which can be used information such as age, sex, geographic location, or even answers provided on a dedicated questionnaire. Content-based strategies require additional information of the items that often are not available or are difficult to collect [32].

Another strategy is based on prior behavior of users without the need to derive a profile for the same [32]. This approach is known as collaborative filtering. There is no need to enrich profiles for each item or user. The idea is to analyze the relationships between users and the interdependencies between items to identify new associations between items and users.

2.1.2.1. Content Filtering

The main idea behind the recommender systems based on content is to suggest items for users that are similar to the items that these users have already positively evaluated sometime before. Recommendations based on content filtering are an

extension of research related to information filtering [33]. In a content-based system, the objects of interest are defined by their characteristics.

In a content-based system, each item must have a profile, which is a record or collection of records representing important characteristics of that item. In simple cases, the profile consists of some characteristics of the item that are easily discovered. For example, the set of actors of a video. Some viewers prefer videos with their favorite actors. Another one could be the director. Some viewers have a preference for the work of certain directors. There are many other features of movies that could be used as well.

Many other classes of items also allow us to obtain features from available data, even if that data must at some point be entered by hand. For instance, products often have descriptions written by the manufacturer, giving features relevant to that class of product. Books have descriptions similar to those for movies, so we can obtain features such as author, year of publication, and genre. Music products such as CD's and MP3 downloads have available features such as artist, composer, and genre.

In formal terms, recommender systems based on content filtering estimate the usefulness $u\ (c, s)$ of an item s to a user c based on utilities $u\ (c, s_i)$ indicated by the user c to items $s_i \in S$ which are similar to the item s [8].

A known successful implementation of content filtering was the Music Genome Project [34], which is used for the Internet radio service Pandora [25]. A trained music analyst scores each song in the Music Genome Project based on hundreds of distinct musical characteristics. These attributes, or genes, capture not only a song's musical identity but also many significant qualities that are relevant to understanding listeners' musical preferences.

An alternative to content filtering relies only on past user behavior—for example, previous transactions or product ratings— without requiring the creation of explicit profiles. This approach is known as collaborative filtering, a term coined by the developers of Tapestry [35], the first recommender system. Collaborative filtering analyzes relationships between users and interdependencies among products to identify new user-item associations.

2.1.2.2. Collaborative Filtering

Collaborative filtering is a technique that uses the history of interactions between users and items aimed at finding relationships between them. An example of relationship could be two videos being viewed by many users, indicating that they are similar. No background information about the item in question is taken into consideration, i.e., the algorithms are not aware of what items or what kind of items are in the system.

In formal terms, recommender systems based on collaborative filtering estimated usefulness value *u (c, s)* of *s* item to a *c* user based on the usefulness *u* (c_j, *s)* of *s* item indicated by the users $c_j \in C$ who are similar to user *c* [8]. For example, in a video recommendation system, to recommend a video *s* to the user *c*, the system will try to find similar c_j users of user *c*, i.e. other users who have the same rated videos in a similar fashion taste. Then, only the videos that are most liked by the c_j users would be recommended to user *c*. This is also known as user-based collaborative filtering.

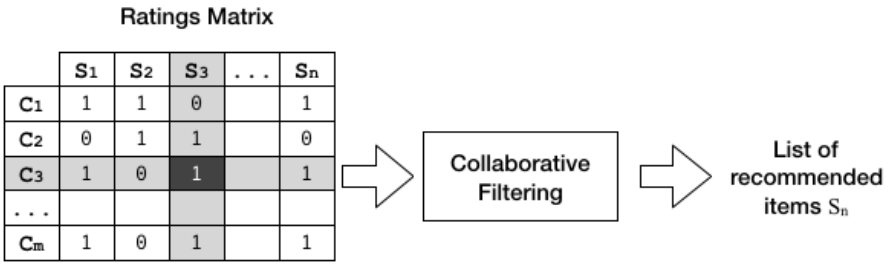

Figure 2 - Collaborative Filtering process

An important characteristic of collaborative filtering is that it is domain free, yet it can address data aspects that are often difficult to profile using content filtering.

It is common in collaborative filtering systems to store user profiles as vectors of items and ratings. The evaluations/ratings from users can be obtained explicitly, or by using some implicit measures. These vectors tend to increase continually as users interact with the system. Some systems take into account temporal dynamics to discount the standard deviations in user interests over time [36, 37]. User feedback may be binary, e.g., liked or not liked, or valued according to preference. Netflix [4], MovieLens [38], and Amazon [3] are among those that adopt the latter approach.

For a detailed survey on item-item recommendation system we suggest the work of Adomavicius and Tuzhilin [8] and the recent book of Rajaraman et al. [9].

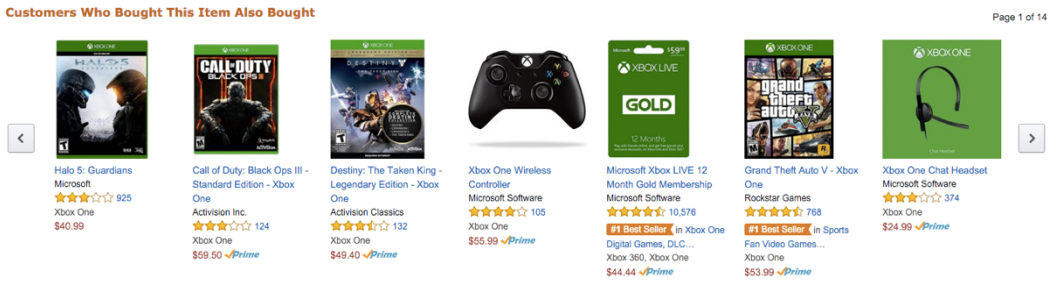

Figure 3 - Collaborative Filtering results in Amazon.com

According to [39], it is possible to group collaborative filtering algorithms into two classes: memory-based (or heuristic-based) and model-based. Memory-based algorithms essentially are heuristics that make evaluation predictions based on the entire collection of previously evaluated items by the users. This approach uses techniques to find a group of users, known as neighbors, that have a history of similar behavior with the target user. Once a neighborhood of users is formed, these systems use different algorithms to combine the preferences of neighbors to produce a prediction or *top-N* recommendation for the active user. These approaches are also known as nearest-neighbor, illustrated in Figure 4 and are the more popular and widely used.

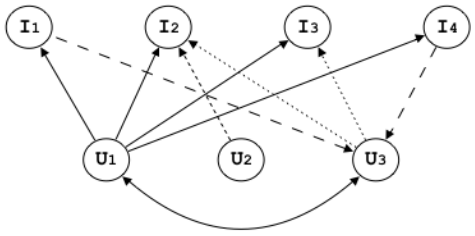

Figure 4 – Neighborhood training process

On the other hand, model-based algorithms use the collection of evaluations to learn a model, which is then used to make predictions. There are many model-based algorithms. These include Bayesian networks, clustering models, latent semantic models such as singular value decomposition, probabilistic latent semantic analysis, multiple multiplicative factor, latent Dirichlet allocation and Markov

decision process based models. For the scope of this thesis, we are interested in the memory-based approaches.

2.1.2.3. Item-based Collaborative Filtering

Different from the user-based collaborative filtering approach, the item-based approach looks into the set of items the target user has rated and computes how similar they are to the target item *s* and then selects *k* most similar items $\{s_1, s_2, \ldots, s_k\}$. At the same time their corresponding similarities $\{sim_1, sim_2, \ldots, sim_k\}$ are also computed. So, one important step in the item-based collaborative filtering approach is to compute the similarity between items and then to select the most similar items.

To compute the similarity between two items s_1 and s_2, the first step is to isolate the users who have rated both of these items and then apply a similarity computation technique to determine the similarity $sim\ s_1, s_2$. Figure 5 illustrates this process, here the matrix rows represent users and the columns represent items. There are several different approaches to compute the similarity between two items. Here are presented three of such methods. These are cosine-based similarity, correlation-based similarity and adjusted-cosine similarity.

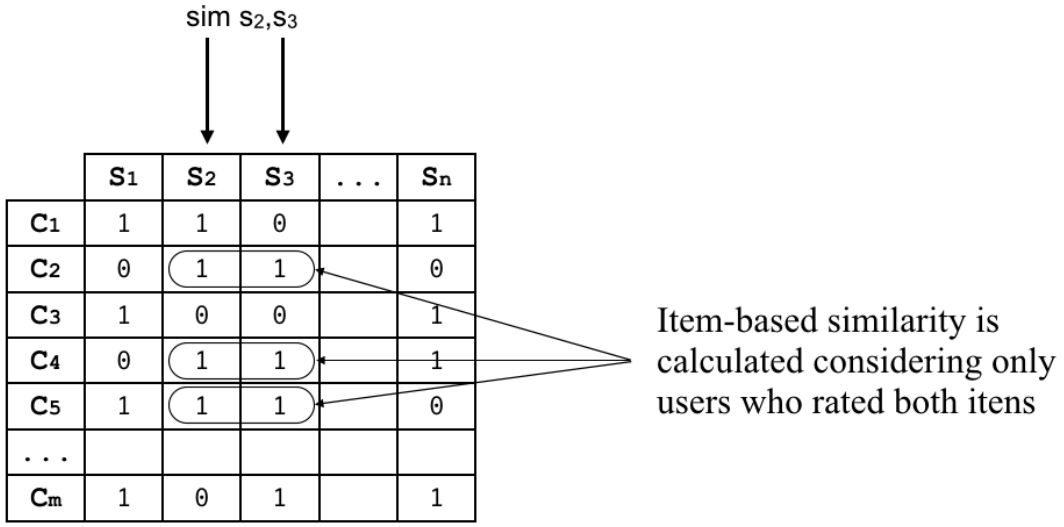

Figure 5 - Item-based collaborative filtering

Cosine-based Similarity

In cosine-based, two items are considered as two vectors in the *M* dimensional user-space. The similarity between those two items is obtained by computing the cosine of the angle between these two vectors. Formally, in the $M \times N$ ratings matrix in Figure 5, similarity between items S_2 and S_3, denoted by $Similarity(S_2, S_3)$ is given by

$$Similarity(S_2, S_3) = \cos\big(\angle(S_2, S_3)\big) = \frac{\langle S_2, S_3 \rangle}{\|S_2\| \,.\, \|S_3\|}$$

where $\angle$, $\langle \ .,. \rangle$ and $\|.\|$ represents, respectively, the angle between the two vectors, the usual inner product in $\mathbb{R}^M$ and the Euclidian vector norm.

Correlation-based Similarity

In the correlation-based, the similarity between two items *i* and *j* is measured by computing the Pearson-r correlation *corr i,j*. To make the correlation computation accurate we must first isolate the co-rated cases (i.e., cases where the users rated both *i* and *j*) as shown in Figure 5. Let the set of users who both rated *i* and *j* are denoted by *U* then the correlation similarity is given by

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_i)(R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_i)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_j)^2}}.$$

Here $R_{u,i}$ denotes the rating of user *u* on item *i*, $\bar{R}_i$ is the average rating of the *i-th* item.

Adjusted Cosine Similarity

One important difference between the similarity computation in user-based collaborative filtering and item-based, is that in case of user-based the similarity is calculated along the rows of the matrix but in case of the item-based the similarity is computed along the columns i.e., each pair in the co-rated set corresponds to a different user (Figure 5). Computing similarity using basic cosine measure in item-based case, as presented previously, has one important drawback: the difference in rating scale between different users are not taken into account. The adjusted cosine

similarity offsets this drawback by subtracting the corresponding user average from each co-rated pair. Formally, the similarity between items *i* and *j* using this scheme is given by

$$im(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_u)(R_{u,j} - \bar{R}_u)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_u)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_u)^2}}.$$

Here $\bar{R}_u$ is the average of the *u-th* user's ratings.

2.1.3. Limitations of Collaborative Filtering

These collaborative filtering methods have been successful in many areas, but its algorithms have some limitations:

1. S*parsity of data*. Neighborhood methods are based on exact matches, making the algorithms sacrifice the accuracy of the recommender systems [14]. Once the correlation coefficient is defined only between users who have rated at least two items in common, many pairs of users have no correlation. In practice, many recommender systems are used to operate large data sets, for example, Amazon.com recommending books and eBay with used products. In these systems, even highly active members may have interacted or assessed far less than 1% of the items. Thus, neighborhood algorithms may be unable to make good items recommendations for a given user.

2. *Scalability*. Neighborhood algorithms require a computational effort that is able to grow with both the number of users and the number of items. With millions of customers and products, a standard web recommendation system running existing algorithms end up facing scalability issues.

3. *Synonyms*. In a real scenario, different product names may refer to similar objects. Recommender systems based on correlation can not find this latent association and treat these items as different products.

There are many challenges for collaborative filtering systems. Its algorithms need to have high performance in sets of sparse data, scale with the increase in the number of users and items, achieving good design recommendations in a short

period of time and still deal with other problems such as noise in the data set and privacy.

2.1.4. Comparison of Strategies

There are advantages and disadvantages to both types of approaches to recommender systems.

The main drawback of the approaches based on content filtering compared to those based on collaborative filtering is that they rely heavily on metadata describing the items, implying that this information must be obtained and entered into the system. Thus, the maintenance of a content-based recommendation system requires an effort significant, since there is a constant need to keep information or add new information when new items are added to the system.

Another issue is that not all information can be easily expressed or consistently. Something as simple as the genre of a film can not be used directly because this information is not always well defined. Users can have different interpretations of the genre to which a film belongs also the same film can be between two distinct genres. Using the genre data can thus cause problems when the system based on this type of information.

Collaborative filtering approaches do not depend on these types of information, which makes them more useful in areas where it is not clear which attributes of the items imply on user preferences. The main deficiency of collaborative filtering algorithms, however, is what is called a cold start problem. Since these algorithms rely on assessment information, it is impossible to recommend items that have not been evaluated by users, or give recommendations to users who have not provided any preference information.

Moreover, since these systems have preference information is necessary for users to share their tastes. Users who are very concerned about the issue of privacy can consider this need to share a problem. In recommender systems based on content, there is no need to share information between users because the algorithms only use the personal evaluations and similarities between users in terms of meta-information, not comparing the standards of evaluation.

Taking these advantages and disadvantages into account the conclusion is that different approaches are most appropriate in different domains. There are attempts to combine collaborative filtering and content-based filtering movie recommender systems to circumvent the shortcomings of each method [30, 40], but because of the ease of maintenance the collaborative filtering algorithms have been most widely used in most recommender systems today.

However, as discussed previously, there are a number o challenges related with implementation of collaborative filtering applications. One of them is the scalability. This thesis argues that the combination of a collaborative filtering approach with cloud computing technologies could tackle such limitations. In the next section the key fundamental concepts of cloud computing used in this thesis are presented.

2.2. Cloud Computing

Today's web is filled with browser-based applications that are used regularly by millions of users. Some applications have to deal with billions of server requests on a daily basis, and these numbers keep growing. Such applications need to be designed to scale, to expand onto improved and/or additional hardware, and to do this transparently (or at the least without having to take down the application for maintenance). The hardware that a web application runs on is an important component when it comes to dealing with large-scale applications.

Applications such as Gmail, YouTube and Flickr that run across thousands of machines are good examples since they require different types of hardware, e.g. web-servers, databases, etc. However, as the number of users grows, the cost to keep the application up and running increases dramatically. The cost for maintenance, physical space, cooling, power, operations, increases for each server that is added to the environment, even if its resources are not fully used.

With very large applications, such as YouTube, that typically have more than 2 billion videos watched in a month, the costs associated with maintaining the required infrastructure is unpredictable. To manage this infrastructure, there are basically two options:

1. To provide resources based on peak situations, which basically means that, for the most part, resources will be idle;
2. Provide resources based on the average number of requests, which means that in some situations the servers will be overloaded and the quality of service will be affected.

None of the above alternatives is good from a business perspective. The first one is very expensive, as the cost is basically associated with the price of hardware itself, and not with its usage. In the second one, the quality of service may be impacted, and, in the long run, it may signify loss of clients and/or business opportunities.

In this scenario Cloud Computing appears as an interesting alternative, as its "everything as a service" model provides an economically attractive solution to demand variations.

2.2.1. Cloud Computing Paradigms

For the purpose of this thesis, Cloud Computing is defined as an Internet-based computing, where there is a large group of interconnected computers (cloud), that share their resources, software, and information (computing), on demand, according to the user needs [41]. Vaquero et al. attempt to pin down a suitable definition for clouds that describes how they differ from grids. Their proposed definition is thorough, but verbose:

"Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs." [41]

In creating their definition, Vaquero et al. studied definitions from numerous experts, which featured many attributes of cloud computing such as immediate scalability, optimal usage of resources, pay-as-you-go pricing models, and virtualized hardware and software.

Cloud computing is a paradigm shift following the shift from mainframe to client–server in the early 1980s. Details are abstracted from the users, who no longer have need expertise in, or control over, the technology infrastructure "in the cloud" that supports them [42]. Cloud computing describes a new supplement, consumption, and delivery model for IT services based on the Internet, and it typically involves over-the-Internet provision of dynamically scalable and often virtualized resources [43, 44].

Cloud computing infrastructures are typically broken down into three layers [41, 45]: “software”, “platform” and “infrastructure”. Each layer serves a different purpose and offers different products to businesses and individuals around the world, and, conversely, every layer can be perceived as a customer of the layer below [45].

The Software as a Service (SaaS) layer offers service based on the concept of renting software from a service provider, rather than buying it yourself. It basically refers to providing on-demand applications over the Internet. The software is hosted on centralized network servers and made available over the web or, also, intranet. Also known as “software on demand” it is currently the most popular type of cloud computing by offering high flexibility, enhanced scalability and less maintenance.

Yahoo mail, Google docs, Flickr, Google Calendar are all instances of SaaS. With a cloud-based email service, for example, all that one has to do is register and login to the central system to start to send and receive messages. The service provider hosts both the application and data, so the end user is free to use the service from anywhere. SaaS is very effective in lowering the costs of business as it provides the business an access to applications at a cost normally less expensive than a licensed application fee. This is only possible due to its monthly fees based revenue model. With SaaS, users no longer need to worry about installation or upgrades [46].

Platform as a Service (PaaS) offers a development platform for developers. End users write their own code and the PaaS provider uploads that code and presents it on the web. SalesForce.com’s is an example of PaaS. PaaS provides services to develop, test, deploy, host and maintain applications in the same integrated development environment. It also provides some level of support for the design of software applications. Thus, PaaS offers a faster and usually cost effective model for software application development and deployment.

The PaaS provider manages upgrades, patches and other routine system maintenance. PaaS is based on a metering or subscription model so users only pay for what they use. Users take what they need without worrying about the complexity behind the scenes [47].

There are basically four types of PaaS solutions – *social application platforms*, *raw compute platforms, web application platforms* and *business application platforms* [48]. Facebook is a type of social application platform wherein third parties can write new applications that are then made available to end users.

The final layer in the cloud computing stack is the infrastructure. Infrastructure as a Service (IaaS) is the process in which computing infrastructure is delivered as a fully outsourced service. Some of the companies that provide infrastructure services are IBM, Amazon.com, among others. Managed hosting and provision of development environments are the services included in the IaaS layer. The user can buy the infrastructure according to his or her requirements at any particular point of time, instead of buying an infrastructure that may not be used for months. IaaS operates on a "Pay as you go" model, ensuring that the users pay for only what they are using.

Virtualization enables IaaS providers to offer almost unlimited instances of servers to customers, and make use of the hosting hardware cost-effective. IaaS users enjoy access to enterprise grade IT Infrastructure and resources, that might be very costly if otherwise purchased. Thus, dynamic scaling, usage based pricing, reduced costs and access to premium IT resources are some of the benefits of IaaS. IaaS is also sometimes referred to as Hardware as a Service (HaaS).

An Infrastructure as a Service offering also provides maximum flexibility because just about anything that can be virtualized can be run on these platforms. This is perhaps the biggest benefit of an IaaS environment. For a startup or small business, one of the most complex things to do is keep capital expenditures under control. By moving the computational infrastructure to the cloud, one has the ability to scale up and down as needed.

In next section, one can find details about the Amazon Web Services Cloud Platform, as well as its main services, that provided the basic infrastructure for the architecture proposed in this thesis.

2.2.2. Amazon Web Services Platform

Amazon Web Services (AWS) [49] is a group of cloud-based services provided by Amazon that differs from traditional hosting since its resources are charged by actual usage. These services provide cloud-based computation, storage and other functionality that enable organizations and individuals to deploy applications and services on an on-demand basis and at commodity prices [45]. The AWS platform is composed by several services that complement one another. Elastic Compute Cloud (EC2), for processing, the Simple Storage Service (S3), for binary storage, Elastic Load Balancer (ELB), for load balancing, Elastic Block Store (EBS), for persistent machine storage, SimpleDB, for structured data storage, Relational Database Service (RDS), for relational databases, Cloud Front, for content delivery, are examples of such services. For the purposes of this project, EC2, ELB and EBS are the most relevant services.

Amazon EC2 is a web service interface that provides resizable computing capacity in the cloud. Based on IaaS model, it allows a complete control of computing resources and reduces the time required to obtain and boot new server instances. Users of EC2 can launch and terminate server instances on a matter of minutes, as opposed to delays of several hours, days or weeks, typical of traditional hardware solutions. This feature is particularly interesting because it allows applications to quickly scale up and down their processing resources, as computing requirements change, while in the traditional hardware approach, it can take several weeks or even months to get a new server running.

Amazon EC2 provides developers with two APIs for interacting with the service, allowing instances administration operations, such as start, stop, reboot, query information, etc. One is the Query API in which operations send data using GET or POST methods over HTTP or HTTPS. The other is the SOAP [50] API in which operations send data using SOAP 1.1 over HTTPS.

The main concept behind EC2 is that of a server instances [51]. There are a number of different types of instances that users can choose from, divided into five categories: general purpose, compute optimized, memory optimized, GPU and storage optimized. Each type has several subtypes, with various levels of processing

power, memory and storage, and users can choose between them according their needs.

Because AWS is built on top of heterogeneous hardware processing power, a standard measure Amazon EC2 Compute Units is used. One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor [52].

The storage provided on an instance (referred to by Amazon as "instance storage") is volatile. Data will survive the instance rebooting, either intentionally or accidentally, but it will not survive an underlying hard drive failing or an instance being terminated. It is also possible to choose a non-volatile storage for the instance, called Elastic Block Storage (EBS). Amazon EBS allows users to create storage volumes from 1 GB to 1 TB that can be mounted as devices by Amazon EC2 instances. Multiple volumes can be mounted to the same instance. Using an EBS as instance storage, users can temporally stop their instances, without data loss.

The EC2 Instances are created by launching machine images known as Amazon Machine Images (AMI) [53], which contain the operating system that will be launched on the instance, along with software applications and their configuration. AMIs are stored on Amazon S3 and Amazon provides a number of pre-bundled public AMIs (with Linux, UNIX or Windows as the OS), that can be immediately launched by users, and do not require specific configuration.

Users can also create their own custom AMIs (private AMIs), either from scratch or using a public AMI as base. Private AMIs are created by a process called bundling, in which a machine image is compressed, encrypted and split, the parts of which are then uploaded to Amazon S3.

EC2 provides the ability to place instances in multiple locations. EC2 locations are composed of Regions and Availability Zones. Regions consist of one or more Availability Zones, are geographically dispersed. Availability Zones are distinct locations that are engineered to be insulated from failures in other Availability Zones and provide inexpensive, low latency network connectivity to other Availability Zones in the same Region [45].

The Amazon S3 service provides a simple web service interface that can be used to store and retrieve data on the web, and provides a scalable data storage

infrastructure [54]. It is designed to make storing and retrieving data on AWS as simple as possible. Data is stored using a straightforward flat model, on top of which users can build their own storage structures using hierarchies. S3 also features a simple, yet versatile, access control system, where objects can be made private, public or be made accessible by certain groups of users.

The two main concepts of S3 are buckets and objects [45]. Buckets are containers for data objects. All objects stored on S3 are stored in buckets. An object consists of four components: a value (the data being stored in that object), a key (the unique identifier for that object), metadata (additional data associated with the object) and an access control policy.

Each bucket has a name that is completely unique within S3. Bucket names are directly mapped to URLs for addressing data stored on S3. If a bucket is named *mybucket* then it can be addressed with the URL *http://mybucket.s3.amazonaws.com*. This URL can be appended with the name of an object to create an address for any object stored on S3.

The other main concept of S3 is an object. Object sizes vary from one byte to five gigabytes. There is no limit to the number of objects that a user can store on S3 and no limit to the number of objects that can be stored in a bucket. A bucket can be stored in one of several Regions. Users can choose a Region to optimize latency, minimize costs, or address regulatory requirements [45, 54].

S3 objects are redundantly stored on multiple devices across multiple facilities in an Amazon S3 Region. To help ensure durability, Amazon S3 PUT and COPY operations synchronously store your data across multiple facilities before returning. Once stored, Amazon S3 maintains the durability of your objects by quickly detecting and repairing any lost redundancy. Amazon S3 also regularly verifies the integrity of data stored using checksums. If corruption is detected, it is repaired using redundant data. In addition, Amazon S3 calculates checksums on all network traffic to detect corruption of data packets when storing or retrieving data [54].

The key is the name of the object and must be absolutely unique within the bucket that contains the object. Keys can be any size from one byte to 1,024 bytes. Keys can be listed by their bucket and a prefix. This allows users to use common prefixes to group together their objects into a hierarchy, meaning that the flat storage

model of S3 buckets can then be turned into a directory-like model for storing data. Object keys can also be given suffixes, like *.jpeg* or *.mpeg*, to help make the key more.

The metadata of an object is a set of key/value pairs and is divided into two types: system metadata and user metadata. System metadata is used by S3 while user metadata can be any key/value pair defined by the user. User metadata keys and values can be any length, as long as the total size of all metadata (system and user) for an object is less than two kilobytes.

Access control on objects is managed by access control lists (ACL). Every object, as well as every bucket, has an ACL. When a request is made to S3, it checks the ACL of the object or bucket to check if the requester has been granted permission. If the requester is not authorized to access the object then an error is returned by S3. There are a number of different types of groups that can be granted permissions and a number of different permissions, such as READ, WRITE and FULL CONTROL.

S3 provides two APIs for making requests, the first uses a REST protocol and the second uses SOAP. The REST API uses standard HTTP headers and status codes, with some additional headers added in by S3 to increase functionality.

Amazon S3 also has the option of the Reduced Redundancy Storage (RRS) that enables customers to reduce their costs by storing non-critical, reproducible data at lower levels of redundancy than Amazon S3's standard storage. It provides a cost-effective solution for distributing or sharing content that is durably stored elsewhere, or for storing thumbnails, transcoded media, or other processed data that can be easily reproduced. The RRS option stores objects on multiple devices across multiple facilities, providing 400 times the durability of a typical disk drive, but does not replicate objects as many times as standard Amazon S3 storage, and thus is even more cost effective [54].

Amazon AWS also offers the Elastic Load Balancing (ELB) service, which automatically distributes incoming application traffic across multiple Amazon EC2 instances in the cloud. It enables the development of applications with greater levels of fault tolerance, seamlessly providing the required amount of load balancing capacity needed to distribute application traffic. Elastic Load Balancing

automatically scales its request handling capacity to meet the demands of application traffic. Additionally, Elastic Load Balancing offers integration with Auto Scaling to ensure that you have back-end capacity to meet varying levels of traffic levels without requiring manual intervention.

The combination of Elastic Load Balancing with Elastic Compute Cloud provides an extremely scalable solution which allows the continuous adaptation of deployed infrastructure according to demand variations. This combination is a fundamental component of the architecture proposed in this thesis, allowing the provisioning of additional computing resources in order the address the needs of large scale collaborative filtering.

The next chapter details why large scale collaborative filtering presents scalability issues, and chapter 4 will present how this cloud computing technologies are used to address those issues.