Data & Knowledge Engineering 116 (2018) 100–123

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

A cloud services recommendation system based on Fuzzy Formal Concept Analysis

Haithem Mezni[a,*], Taher Abdeljaoued[b]

[a] SMART Lab, University of Jendouba, Tunisia
[b] SMART Lab, University of Tunis, Tunisia

ARTICLE INFO

Keywords:
Cloud computing
Recommender system
Cloud service recommendation
Collaborative filtering
Fuzzy Formal Concept Analysis

ABSTRACT

Cloud computing is an attractive paradigm which offers variant services on demand. Many available cloud services offer the same or similar functionalities, which made it challenging for cloud users to choose a suitable service that meets with their preferences. Existing service selection approaches were not enough to solve this challenge. That's why researchers went for recommendation approaches trying to find a solution. Cloud service recommendation has become an important technique for cloud services. It helps users decide whether a service satisfies their requirements or not. However, two main recommendation problems remain unsolved yet, data sparsity and cold start. In addition, existing solutions mostly tried to adapt techniques inherited from Web service and e-commerce domains. This approach is not always adequate due to many reasons such as the cloud architecture, the various service models, etc. To address the problems stated above, we propose a *Collaborative Filtering* based recommendation system for cloud services using Fuzzy Formal Concept Analysis (Fuzzy FCA). Fuzzy FCA has a solid mathematical foundation and it's based on the *lattice theory*. The lattice representation will give an explicit description of our cloud environment (users, services, ratings, etc.) and, then, extract the pertinent information from it (similar users to an active user, ratings of each similar user, top services, etc.) which will make the recommendations more suitable. Experimental results confirmed our expectations and proved the efficiency of such an approach.

1. Introduction

Cloud computing is an emerging technology that offers services on demand (computing resources, storage resources, applications, etc.) [1,2]. It is becoming more and more popular today, since it provides a lot of positive characteristics that encourage Web users to try it, such as reliability, security, agility, performance, etc. The most important feature that cloud came up with is the pay-as-you-go aspect [1]. Users don't have to buy whole applications just for a brief usage. They just pay the time spent on using a certain service of the cloud. Elasticity is another important feature offered by the cloud [1] as it allows to scale computing capacity up or down on-demand, to allocate exactly the resources needed by a service.

But, with the increasing number of available cloud services nowadays, a lot of services offer the same or similar functionalities which made it challenging for the users to choose a suitable service that meets exactly with their requirements [2]. Traditional selection approaches suggested differentiating cloud services by evaluating some important criteria such as QoS (e.g., response time, security, availability, reliability, throughput, etc.), which are non-functional attributes [2]. However, the number of services that may satisfy user needs with a high QoS level remains huge. Consequently, these services must be differentiated by additional criteria other

* Corresponding author. Campus Univ. Avenue UMA, 8198 Jendouba, Tunisia.
E-mail address: haithem.mezni@gmail.com (H. Mezni).

https://doi.org/10.1016/j.datak.2018.05.008
Received 6 October 2017; Received in revised form 25 March 2018; Accepted 30 May 2018
Available online 01 June 2018
0169-023X/ © 2018 Elsevier B.V. All rights reserved.

than QoS, such as users ratings, service execution contexts, providers' policies, pricing models, etc. For example, a cloud service may be recommended based on the availability zone of the user, while other high quality services are excluded because they don't offer a suitable pricing model. Existing Cloud service selection approaches are not able to deliver suitable services because they mainly focus on QoS constraints and often ignore other useful information, such as the services' usage history, the user's neighborhood, etc. A more effective solution for the problem stated above would be the use of a *recommendation system*.

Not a lot of works have been published about the cloud recommendation challenge since it's still a new research area, but the existing papers tried to address this problem with different solutions. However, two main cloud service recommendation challenges remain unsolved still, *data sparsity* and *cold start*. Cold start problem refers to the lack of information on new items or new users. This situation arises when a new user has not rated a sufficient number of items, or a new item has not been rated by a substantial number of users. In this case, the recommendation algorithm will not find a good neighborhood for the user or the item, in order to effectively predict their scores, and will be unable to offer reliable and accurate recommendations [55].

Data sparsity refers to the situation when available data (e.g., ratings) is not sufficient to determine the neighborhood of the active user (i.e. similar users). This problem mainly arises when the user-item matrix used for recommendation is extremely sparse (a big number of users and items, but with few ratings) [55].

In the context of cloud service recommendation, the cold start problem is when new cloud users don't use new services. Those services will not have any ratings, therefore will not be recommended to other users. Since most of the recommendation systems are based on nearest neighbors' information, new users will not have any relation with similar users. Data sparseness is a problem that occurs when there is many services/items to rate [3]. Most of the services will not have accurate ratings or any ratings at all. A lot of the available services will not be used for that reason.

Trying to address the challenges stated above, we adopted a Collaborative Filtering (CF) by means of Fuzzy Formal Concept Analysis (Fuzzy FCA) approach in our work. CF methods are the most suited for service recommendation problems due to their effectiveness [4]. On the other hand, Fuzzy FCA is a formal technique for data analysis that deals with uncertain and vague information [49]. It offers a valued representation (also called truth degree) of the object/attribute relations in [0,1]. The mathematical foundation of Fuzzy FCA [5] method ensures a high level of accuracy and trust in the recommendations generation. Based on the lattice representation, it gives an explicit description of the objects of the cloud system environment (users, services, ratings), which makes the recommendations more suitable for the targeted user. Fuzzy FCA was applied with success in several domains, particularly in Web services for service selection [28,29].

Fuzzy FCA has several strengths. First, it allows for hierarchical and weighted representation of information into small clusters called *fuzzy formal concepts*, which facilitates access to the most pertinent information. Second, it allows excluding useless data if objects do not satisfy specific properties. Third, it allows for a top-down or a bottom-up parsing of the lattice structure in order to find pertinent data in the minimum set of formal concepts, which has a positive impact on the processing time. In our approach, we exploited these strengths to organize the usage history (QoS, ratings, neighborhood) of cloud services into small clusters, depending on the *user-service-rating* relations. Also, by adopting the principle of truth degree (or confidence threshold) offered by Fuzzy FCA, all of the relations *user-service* (i.e. ratings values), *user-user* (i.e. similarity degrees), *service-capability* (i.e. QoS levels), which have a confidence value less than a specified threshold will not be shown in the lattices' hierarchies. This means that the recommendation process will be limited, not only to the Cloud services with high QoS levels and top ratings, but also to the most similar users. Regarding the lattice parsing methods offered by Fuzzy FCA, they will be very useful to rapidly extract the sub-hierarchy of similar users and the top-rated services that almost satisfy the user requirements and high QoS levels.

Fuzzy FCA may be a suitable solution to the problems of data sparseness and cold start. The fuzzy lattice representation of services' and users' data allows to evaluate how much the users' ratings and neighborhood will be sufficient to produce pertinent recommendations, even in case of sparse formal context and in case of new users or few ratings on a cloud service. Also, Fuzzy FCA allows excluding useless data (e.g., unrated or poorly rated services) which will reduce the amount of incomplete or uncertain data and increases the density of the ratings formal context, which solves the data sparseness problem. Regarding the cold start problem, if new users' information matches few features in the available ratings and similarities with other users, the new users together with their data will be partially considered and the similarity calculation will be performed to find a better neighborhood for the service, based on the few available users in the formal concepts. Also, using the Galois sub-hierarchy (GSH) allows to reduce the size of the fuzzy lattice and limits the recommendation to the users that positively evaluated Cloud services at least one time, unlike the existing approaches that exclude the new users or users/services with few ratings.

Our main contributions are summarized as follows:

- Cloud environment modelling using Fuzzy FCA. Cloud services information, relations between users and ratings will be modelled using the lattice representation. That will ensure pertinent services selection and suitable recommendations.
- Definition of reliable information extraction algorithms that run through the cloud environment lattices to select candidate cloud services and return the best rated services (Top_K) according to the active user's neighborhood.
- Implementing the Fuzzy FCA-based service recommendation approach and experimentally evaluating the performance and the quality of produced recommendations, in comparison with existing state-of-the-art solutions.

The rest of the paper is organized as follows: Section 2 presents the Cloud computing service delivery model and defines the problem of Cloud service recommendation. Section 3 presents the main classes of the used approaches for the cloud recommendation problem. It also briefly discusses some of their limits and gaps and how we addressed those limitations with our proposed approach. Section 4 presents Fuzzy FCA and the reasons of its adoption in our work. Section 5 presents our contributions and the way we

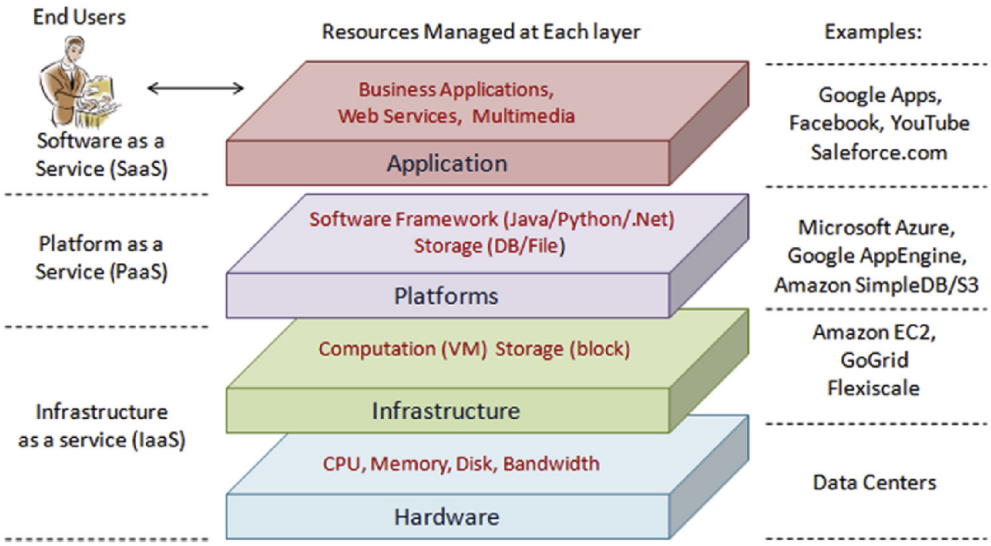

Fig. 1. Cloud computing architecture [10].

adopted Fuzzy FCA for the cloud services recommendation. Section 6 presents a theoretical complexity study of our proposed approach. A recommendation scenario is illustrated in Section 7 based on IaaS cloud service model. A set of conducted experiments and a discussion of the results are presented in Section 8. The last section concludes our work and discusses some of the future works and the possible enhancements.

2. Background

2.1. Cloud computing

Since its emergence as a new service provisioning model, Cloud computing promises several attractive benefits for businesses and users. Two of the main benefits include [2]: *elasticity*, which is the ability to scale computing capacity up or down on-demand; and *pay-as-you-go*, that allows users pay only for the resources and workloads they need.

Cloud computing offers multiple types of services, which are divided mainly into three groups: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). These service models may have a relation between each other. For example, PaaS is dependent on IaaS because application platforms require physical infrastructure (see Fig. 1).

- *IaaS:* provides fundamental computing resources to customers that include storage, computation capacity, load balancers, network devices and firewalls. Users have an allocated storage capacity and can start, stop, access and configure the VMs and storage as desired. Examples of IaaS services include Amazon Web Services (AWS), Cisco Metapod, Microsoft Azure, Google Compute Engine (GCE), Joyent [2].
- *PaaS:* represents a development platform that supports the full life-cycle of a software. It allows cloud users to develop applications and services. Additionally, it offers deployment and configuration management tools for cloud applications. PaaS is used for general software development and many PaaS providers will host the software after its developed. Google AppEngine and Apprenda [2] are examples of PaaS services.
- *SaaS:* offers cloud customers the ability to access and utilize applications that are offered by cloud providers across the Internet. This software deployment and distribution model allows to deliver software applications as Web services. SaaS examples include Google Apps, Salesforce, Workday, Concur, Citrix GoToMeeting, Cisco WebEx. [2].

Besides these three shapes of Cloud services, other service models have emerged. We cite Hardware as a Service (HaaS), Framework as a Service (FaaS), Database as a service (DBaaS), Network as a Service (NaaS), Business process as a Service (BPaaS), etc. [12].

2.2. Cloud servcie selection and recommendation

The amount of Cloud services together with their related information available on the Cloud has become immense and is still growing at a fast rate. In addition, the emergence of the social networks (e.g., Facebook, Twitter) and mobile cloud environments has boosted the volume of online information resources, since these technologies enable information sharing and offer various types of Cloud services such as storage, computation, etc. This makes it difficult for Cloud consumers to find useful service instances, which is a problem known as the information overload [31].

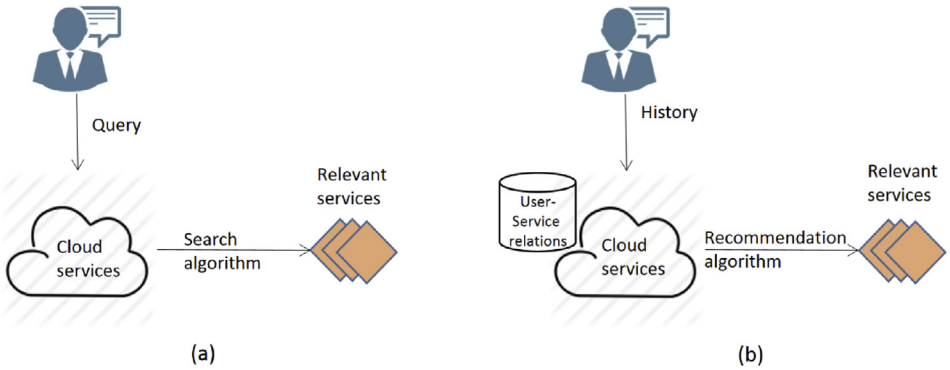

Fig. 2. (a) Cloud search engine (b) Cloud recommender system.

In the literature, two major techniques are adopted to help users deal with the problem stated above and to select suitable Cloud services: *search engines* and *recommendation systems*. In the first technique, as shown in Fig. 2 (a), a cloud consumer submits a simple query about the required cloud service to the search engine. The system, then, tries to find the desired service (e.g., VM, storage server, SaaS service) in the collection that best matches the query submitted. For example, a cloud user that looks for a computation environment may specify his requirements in terms of needed resources (e.g., CPU, RAM, storage), required latency and performance levels, size of VM instance, etc.

In the recommendation case, as shown in Fig. 2 (b), the outputs of the search phase (selected cloud services) are considered as inputs to the recommender system, and the user needs are expressed implicitly (e.g., based on the past experience or the cloud availability zone of the user). Hence, the top-rated services are determined by using a filtering technique, such as content-based filtering, collaborative filtering or hybrid filtering [31]. In this case, the real QoS levels of cloud services are derived from the history of similar users and/or similar services.

Cloud service selection is the process of determining the most suitable service that meets at best the user needs and preferences [40]. For example, for a SaaS provider that wants to deploy its SaaS application and data components in the cloud, the optimal availability zone that offers the most performant servers must be selected based on various criteria (e.g., CPU processing capacity, RAM, storage, etc.), in order to guarantee an optimal SaaS placement. Cloud services may also be selected depending on other requirements including, cost, latency, security, providers' policies, pricing models, etc.

To distinguish between the huge number of similar Cloud services, Quality-of-Service (QoS) is considered as a key factor in the service selection and recommendation process. Considered as non-functional requirements, QoS properties are additional information used to evaluate the degree that a service meets specified quality requirements in a user request [55]. The QoS could be classified as dynamic QoS (e.g., Performance, Response time, Reliability, Availability) and static QoS (e.g., Robustness, Accuracy, Security). Each one has a dimension and can be measured and evaluated using one or more QoS metrics. QoS attributes could be in the form of values sent by providers (e.g., price), user's feedback (e.g., reputation) or third parties (e.g., response time). In cloud computing, the QoS depends on the cloud service model. Taking as example the QoS attributes specific to IaaS, such as Memory, CPU capacity, Accuracy, VM response, Throughput, Storage, Stable time, Failure rate, etc. QoS attributes for a PaaS services include Availability, Accessibility, Security, Reliability, etc. Whereas, those specific to SaaS include Security, Reliability, Performance, Interoperability, Scalability, Availability, etc., (Burkon, 2013). QoS attributes are generally expressed in the user request, depending on the demanded cloud service model.

Although cloud service selection allows filtering available services according to non-functional properties, the number of services that may satisfy user needs with a high QoS level remains huge. In such situation, the recommendation of best services could be considered as a next natural step to refine the services set that results from the selection phase. To meet this goal, new criteria other than QoS must be considered to filter the candidate services, such as users' ratings, service execution environments, etc. For example, selected services may be refined according to the availability zone of the user. In fact, cloud providers may offer several availability zones (physically isolated data centers) in different geographies to support various users' needs. Characteristics of Cloud availability zones include hardware infrastructures and generations, software technologies, placement policies of service instances, zone size, type of offered instances and their prices, etc. [37]. For example, Cluster GPU instances are offered by EC2 in North Virginia zone, while they are unavailable in Sao Paulo zone [37]. This difference in cloud zones' characteristics results in different levels of performance and reliability, as reported in Ref. [38] which indicated variation up to 280% between instances of same type hosted on different processor architectures in EC2. Service instances can be deployed in several availability zones of different Cloud providers. As a result, the quality of user experience (QoE) on a consumed service will vary depending on the hosting Cloud zone. So in such situation, the existing search and selection frameworks, which are mainly based on QoS criteria, will not be able to filter the candidate services, as different instances that satisfy the non-functional requirements and offer high quality levels may not be suitable to the user context. That is why this task must be delegated to recommender systems, in order to deliver optimal services by predicting the user preferences based on various implicit and/or explicit information (e.g., usage history, ratings, Cloud availability zone, providers' policies, etc.).

3. Related work

Back in the 1990s, recommendation systems were first used in *E-commerce* and *Web services* [6,7]. Then, they were adopted in *social networks* [8] and *service selection and composition* [9] to address different challenges. The few papers that have been published to address this problem in the context of cloud computing are mainly based on three classes of approaches: *Content-based Filtering* [3], *Collaborative Filtering* (CF) [11], and *Hybrid Filtering* [6,36].

3.1. Content-based approaches

The content-based approaches are mainly used in the Web service domain. That explains why only very few papers were found that address the cloud service recommendation challenge using a content-based filtering. They are based on finding similar services in terms of characteristics of the services previously used by the active user [3].

Rodrigues et al. [13] proposed a files recommendation engine in a cloud storage environment. The proposed system calculates the similarity between available public files stored on the cloud environment and the files that were previously used by the cloud user. Then, five cloud factors (availability, download rate, similarity, amount of downloads and files size) are applied. If a similar file to the files previously used does not meet with the factors, it will not be recommended.

Content-based filtering is the least used in cloud recommendation. In fact, content-based approaches take into consideration the information related to the services only, to find the similar ones. Also, a lot of services have missing information which will affect the quality of the recommendations. On the other hand, a lot of cloud services are still unused. So, they will not be considered at all in the content-based recommendation process.

3.2. Collaborative filtering based approaches

CF-based approaches identify the neighbors of the *active user* and predict the missing values of the user-service matrix based on the previous behavior of his similar users [4]. CF methods are based on two main approaches:

- *Model based approaches*: provide QoS values prediction by a user preferences model. Several probabilistic and learning algorithms are used for the model creation (K-means clustering, latent factor, etc.) [14]. Another popular way to represent a user preferences model is matrix factorization methods, which are used to detect the implicit information [15].
- *User based approaches*: predict missing QoS values based on the similarity of two users.

3.2.1. Model-based approaches

QoS must be taken into consideration for cloud recommendation, as it is an important feature for cloud users. Cheng et al. [16] proposed an algorithm for user-oriented cloud service recommendation based on QoS prediction. The proposed algorithm calculates the similarities between users and between services based on Pearson correlation coefficient (PCC), the top-k selection and the new user problem. However, only response time was considered as a prediction object.

Zheng et al. [17] proposed a ranking-based approach to recommend cloud services. The approach predicts both QoS ratings and rankings for cloud services. Spearman coefficient is adopted to calculate the similarities between users, while a k-nearest neighbor technique is used to find a set of similar neighbors to the active user. Finally, a CF algorithm is adopted to predict the QoS of the active user for an unrated service, and determine a set of recommended cloud services.

The authors, in Ref. [18], aim to recommend trustworthy cloud services based on dynamic fuzzy clustering. They proposed a preferences-aware trustworthy service recommendation model that consists of two steps: evaluation of cloud services and recommendation of trustworthy services. A set of trust attributes is defined and used to evaluate the trust level of cloud services according to the feedback from consumers and their preferences (usage, trust and cost) in different application scenarios. Based on the fuzzy similarity computation that takes user preferences into account, the dynamic fuzzy clusters are generated, from which the suitable services are recommended.

Even though many QoS models are proposed, measuring QoS attributes is still a difficult task. Only few QoS attributes were taken into consideration leaving behind important ones.

3.2.2. User-based approaches

User-based recommendation is an effective approach for exploiting users preferences to generate recommendations. However, one of the main challenges for user-based methods is data sparsity, since many users have only used a limited number of cloud services. As a result, the recommendation system will not be able to generate accurate recommendations.

Trying to address the challenge stated above, Yu [19] proposed CloudRec, a framework for personalized QoS assessment of cloud services. A Regularized Posterior Probabilistic Nonnegative Matrix Factorization (RPPNMF) was used to capture the cloud services features and use them to regroup the similar users into a set of communities. Those communities will then be analyzed by using an iterative algorithm to predict users QoS perception on unknown cloud services.

Gui et al. [4] came up with a recommendation mechanism to select the best IaaS and PaaS services. First, to collect and unify heterogeneous information that come from different cloud providers, a unified cloud information model was designed to describe the different information (pricing, basic features, user feedback, etc.). Second, an operational service classification model and an application-dependent schema for requirements description are defined. To better evaluate the generated solutions, a user preference-

aware and dynamic changeable solution's evaluation mode is proposed.

Amin et al. [22] proposed an energy-aware recommendation system for cloud infrastructures. By performing offline profiling of cloud nodes, the system generates energy-aware profiles of cloud infrastructures. The online process, however, performs real-time matching between offline generated profiles and real-time usage data. This aims to increase the system's profile creation accuracy, which will later be used in recommendations. Depending on the generated profiles, designed usage logs are used to generate energy-efficient profiles recommendations.

Even though a lot of CF based approaches were proposed, having a lot of malicious users' feedback, missing users ratings and a huge amount of unused services made it hard for CF based recommendation systems to generate pertinent and accurate recommendations to users.

3.3. Hybrid approaches

A hybrid recommendation method uses collaborative filtering and content-based filtering principles. Combining those two can achieve higher prediction accuracy than a single approach [6].

In Ref. [21], Afify et al. propose a personalized recommender system for SaaS services called SaaSRec. To find the relevant business-oriented services, SaaSRec semantically processes SaaS characteristics (QoS and reputation) against user information (physical location, user interests). To calculate the reputation of each candidate service, the authors defined a method based on the objective time-weighted user feedbacks.

Wheal and Yang [23] developed a cloud service identifier, a search engine and a recommender system called CSRecommender. The proposed system runs a CF based algorithm to extract all of the similar users to the active user, then runs a content-based algorithm to extract all of the similar services to the previously used services. However, those two steps are time-consuming.

Yan et al. [24] proposed Monsoon, a recommendation framework that takes into consideration business requirements and company policies and standards. A Policy Implementation Engine (PIE) was designed, containing all of the company policies and the government regulations, to detect conflicts based on user requirements. PIE generates all feasible solutions that satisfy user preferences and company policies. Then, a Cloud Intelligence module analyzes those solutions, taking into consideration several criteria (user's rating and feedback, social media…).

Pereira et al. [15] proposed combining real-time collaborative filtering with the item-item recommendations, to find a set of similar items to the considered one and then predict how would the user evaluate an unrated item to recommend those items. The prediction process takes into consideration the user's explicit and implicit feedback, which will be stored in a multidimensional vector that represents the user-item relation. The similarity is then calculated between the multidimensional vectors representing the items. The similarity is recalculated whenever there is a new entry to the matrix, since the item relevance changes over time in the real world.

Meng et al. [25] proposed a hybrid collaborative time-aware approach, which is divided into two main steps. The first one consists in extracting the neighborhood of both users and items (similar users and similar items) using a temporal-aware neighborhood model. The second step is the use of a temporal-aware latent factor. During this step, both users and services will be characterized by vectors of factors. Ratings will be extracted from the interaction of both users and services vectors. Then, both models are combined by the Simple Additive Weighting (SAW) technique to get the recommendations.

The hybrid filtering seems to be the answer to the drawbacks of the collaborative filtering and the content-based filtering. However, combining those two in one recommendation process is a complex task and will be more time-consuming.

3.4. Discussion

In this subsection, we used different criteria to give a global overview of the existing recommendation approaches, as detailed below:

- *Recommendation technique:* represents the filtering method used in authors' approach, which can be CF, CB or Hybrid.
- *Context:* denotes the cloud model focused on in the recommendation process (e.g., SaaS, IaaS, PaaS).
- *Focused parameter:* represents the recommendation criteria considered by the authors, such as user trust, QoS, etc.

As shown in Table 1, most of the approaches adopted a collaborative filtering. Collaborative filtering was proved to be the most effective approach to solve recommendation issues, because it takes into consideration the past behavior of a targeted user's neighbors. According to his neighbors' history of cloud service usage, the active user will have an idea about whether a service is worth trying or not. The context of the recommender systems is general (recommends IaaS, PaaS and SaaS services all together) in most of the approaches, like Yu et al. [19].

Only a few dealt with a certain cloud model (IaaS, SaaS or PaaS) alone like Zhang et al. [42], Soltani et al. [40], etc. Regarding the parameters that were taken into account in the published works, some authors found it important to focus on the QoS like Cheng et al. [16] and Jayapriya et al. [43]. Others thought that the trust feature is more important, such as Maa et al. [39] and Taneja et al. [41]. Other approaches took into consideration the users history only to generate recommendations, like Zhang et al. [42].

To position our work among the existing literature approaches, we proposed a lattice-graph based classification (see Fig. 3), which allowed us to regroup approaches according to their common features, and identify their major limitations. The approaches lattice consists of a set of clusters called *formal concepts* (gray rectangles). Each concept contains an *Extent* section E and an *Intent* section I. *E*

Table 1
Classification of Cloud service recommendation approaches.

Ref.	Approach	Used Technique	Context	Focused Parameter
[37]	Hua et al. (2012)	CF	General	Trust
[24]	Yan et al. (2012)	Hybrid filtering	Enterprises	Company policies
[15]	Pereira et al. (2013)	Hybrid filtering	General	Real-time processing
[13]	Rodrigues et al. (2013)	Content-based filtering	Files	Cloud Factors
[16]	Cheng et al. (2013)	CF	General	QoS
[38]	Li et al. (2013)	CF	General	User feedback
[17]	Zheng et al. (2017)	CF	General	QoS
[39]	Maa et al. (2014)	CF	General	Trust
[18]	Ma and Hu (2015)	CF	General	QoS & trust
[19]	Yu (2014)	CF	General	RPPNMF
[4]	Gui et al. (2014)	CF	IaaS-PaaS	N/A
[40]	Soltani et al. (2014)	CF	IaaS	N/A
[41]	Taneja et al. (2015)	CF	General	Trust
[23]	Wheal et al. (2015)	Hybrid filtering	General	N/A
[42]	Zhang et al. (2015)	CF	IaaS	User history
[43]	Jayapriya et al. (2015)	CF	General	QoS
[22]	Amin et al. (2015)	CF	IaaS	Energy-aware
[21]	Afify et al. (2016)	Hybrid filtering	SaaS	Reputation & QoS
[25]	Meng et al. (2016)	Hybrid filtering	General	Time

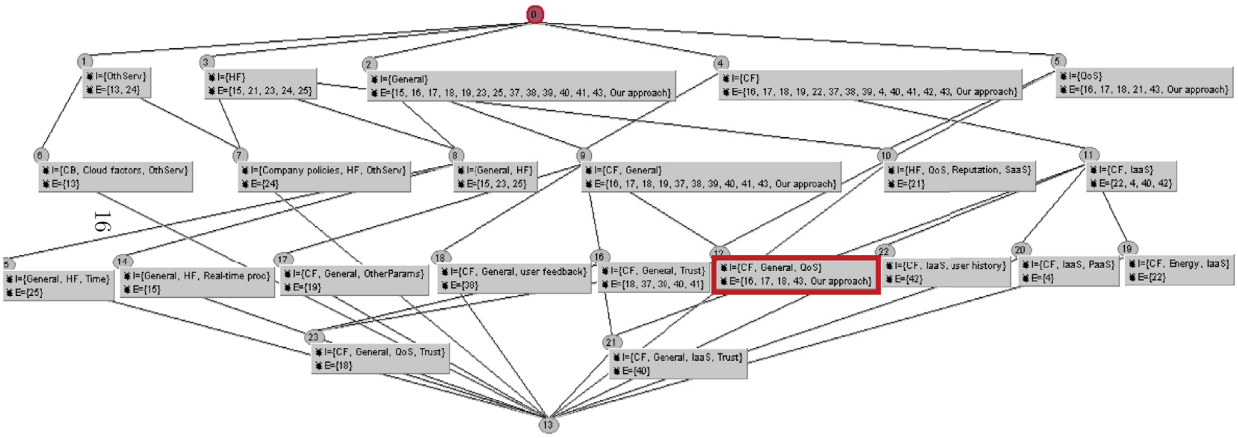

Fig. 3. Lattice of cloud services recommendation approaches.

represents a subset of the existing works on cloud recommendation, as shown in the classification table, whereas *I* represents the common features of the approaches in the Extent section *E*.

From Fig. 3, we can see that four works ([4,22,40,42]) have applied a CF approach to recommend IaaS services only (see concept 11). The formal concept 12 shows that few approaches share some features with our Fuzzy FCA based approach. Indeed, the proposed works in Refs. [16–18,43] adopted CF to recommend cloud services, regardless of their nature and taking into account QoS criteria. However, the authors based their works on the prediction of the QoS values in the generation of recommendations, and they don't offer a rich representation of Cloud services characteristics.

Clearly the collaborative filtering is the most used (see concept 4), followed by the hybrid filtering, which is adopted by five approaches (see concept 3). The general context (SaaS, IaaS, PaaS) in the could recommendation process is the most worked on as well (see concept 2), which raises a question about the ability of existing cloud recommender systems to provide an explicit knowledge representation for a specific cloud service model. The IaaS model is the most dealt with (see concept 11). As for the focused parameters, the QoS takes the first place (see concept 5).

3.5. Requirements for effective cloud service recommendation

From the above discussion, it is clear that most approaches tried to adapt recommendation techniques proposed in Web service and e-commerce domains and, consequently, inherited their limitations. Existing recommendation approaches are fragmented and the focused recommendation criteria limit their adoption and make them inadequate to specific Cloud service models. Also, the majority of the approaches suffer from data sparsity and cold start problems. Other problems are related each to the used method or technique, like ignoring the dependencies between QoS values. For the calculation of similarities between users, reducing the number of similar users or service search domain could present some difficulties for other approaches, where the computation of similarity is

complicated and the choice of the number of similar neighbors is critical.

Apart from the classical cold start and data sparsity problems that have not been solved yet, most approaches address cloud service recommendation without specifying how configurable their recommendation models. They do not allow for a rich and weighted representation of the relations between cloud environment objects (e.g., users, services), as they only adopt a simple matrix representation of services ratings and QoS information. Add to that, these approaches either they adopt a collaborative filtering or a content-based filtering. Even the few works that adopted a hybrid approach, address the collaborative filtering and the content based filtering as two separate techniques that are invoked independently from each other. Moreover, current solutions consider recommendation as a prediction-based problem, and do not allow for effective analysis of the relations between the cloud objects.

Unlike existing works, adopting Fuzzy FCA in our recommendation approach allows to explicitly represent the heterogeneous information in the cloud environment, regardless of the Cloud service model and the recommendation criteria. These information can be organized through a set of formal contexts that may be processed separately or together, depending on the types of requested services. In addition, the fuzzy extension of FCA will permit to organize the *user-service-rating* relations in a weighted hierarchical structure thanks to the lattice theory. This hierarchical representation will help to analyze the hidden relations between users, in order to determine the similarities based on the neighborhood and the consumed services. Moreover, it allows to group users and services into small clusters according to the degrees of similarities, in terms of consumed and best rated services, or in terms of similar services. Hence, similarities between users and between services may together be considered as inputs to the final decision step, which permits to achieve the goal of recommendation.

In the next sections, we present the basic notions of Fuzzy FCA and we detail the proposed recommendation approach.

4. Fuzzy Formal Concept Analysis

Fuzzy FCA [5] is an extension of Formal Concept Analysis (FCA). This latter was first introduced in the early 1980s by Rudolf Wille [26]. It is a mathematical framework based on lattice theory which is exploited to analyze, represent and visualize data using formal contexts. In our work, for a cloud user looking for a cloud service that meets with his requirements, FCA only offers a binary rating of a service. That will not be as significant as a rating in [0,1] for a user. To allow a rating between 0 and 1, Fuzzy FCA can be a good settlement regarding this issue, since it offers a valued representation of the object/attribute relations called *truth degree* (a value in [0,1]). Pioneer studies have integrated FCA with fuzzy techniques to deal with uncertain and vague information.

The basic structures of Fuzzy FCA are presented in the next sub-sections.

4.1. Fuzzy formal context

A Fuzzy formal context is a triplet $K(O, A, R)$, where O is the set of objects, A is the set of attributes and R is a fuzzy relation between an object o and an attribute a, with $o \in O$ and $a \in A$. Each couple (o, a) has a truth degree denoted r, with $r(o, a)$ in [0, 1].

An example of fuzzy relations between objects and attributes is depicted in Fig. 4.

Fig. 4 shows that the object O_1 has all the attributes $A_1, A_2, A_3 \ldots, A_6$ with different truth degree values {0.61, 0.8, 1, …, 0.68}. According to those values, O_1 is more satisfied with the attribute A_3 since it has the highest truth degree value.

The fuzzy formal context allows the identification of the fuzzy formal concepts which will be ordered to construct a hierarchical structure called *fuzzy lattice*, as we will see in the next sub-section.

4.2. Fuzzy lattice

A fuzzy Lattice of a fuzzy formal context K is a set of fuzzy concepts of K. A fuzzy formal concept is a pair (E, I), where E is a set of objects called *Extent* and I is a set of attributes called *Intent*, and all the objects in E are in relation R with all the attributes in I.

- $E = \{o \in O, \forall i \in I, (o, i) \in R\}$
- $I = \{a \in A, \forall e \in E, (e, a) \in R\}$

All objects in the *Extent* of a concept share a set of common attributes of I, and all the attributes of I are shared by the objects of E.

The fuzzy concepts are ordered according to the relation $\leq$ with a confidence threshold denoted T. Any relations with

Example

A	B	C	D	E	F	G
Example	A1	A2	A3	A4	A5	A6
O1	[0.61]	[0.8]	[1]	[0.7]	[0.66]	[0.68]
O2	[0.61]	[0.1]	[0.5]	0	[0.64]	[0.4]
O3	[0.5]	[0.71]	[0.2]	[0.7]	[0.8]	[0.8]
O4	[0.2]	[0.7]	[0.7]	[0.3]	[0.66]	0
O5	[0.1]	[0.5]	[0.4]	0	[0.7]	[0.9]

Fig. 4. Example of a fuzzy formal context.

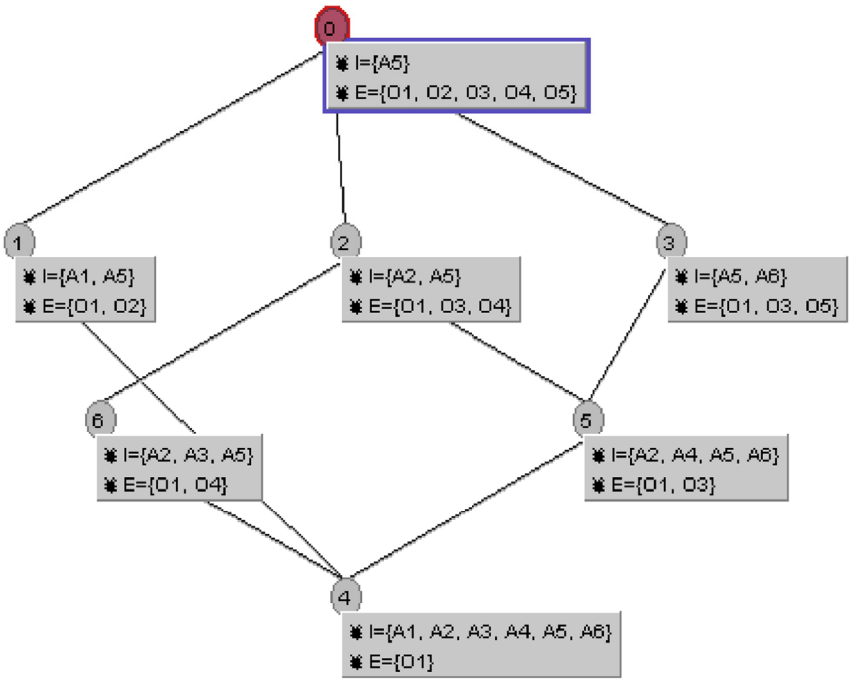

Fig. 5. Example of a fuzzy lattice.

membership values (or trust degree) less than a given threshold will not be shown in the lattice.

In the literature, a variation of algorithms are proposed to extract concept lattices from a given formal context, such as *Bordat*, *Next Neighbor*, *Next Intersection*, *Ganter Next Closure*, ect. [27].

In Fig. 5 below, a fuzzy lattice representation of the previous fuzzy formal context is generated with a threshold $T = 0.6$.

As shown in Fig. 5, all the relations objects/attributes with truth degrees less than 0.6 are not shown in this fuzzy lattice. For example, in the concept C_3, the object O_2 is not found in the Extent section because the membership value of O_2/A_6 is $(0.4) \leq (0.6)$.

4.3. Applying fuzzy FCA in cloud recommendation

Fuzzy FCA has been exploited to solve various research problems including information and resources retrieval [44,45], semantic Web search [46], ontology merging [47], role-based access control [48], etc. In the field of service computing, Fuzzy FCA has been successfully applied in Web service selection [28,29].

Recently, authors adopted FCA and its fuzzy extension trying to address some Cloud computing issues such as Virtual machine scheduling [50], requirements elicitation for cloud services [51] and service discovery [52]. This promising application in the few existing cloud computing approaches, apart from their limitations, suggests that Fuzzy FCA may be a good candidate to the Cloud service recommendation problem.

Another motivation behind using Fuzzy FCA in the cloud service recommendation is its mathematical foundations and soundness. Moreover, Fuzzy FCA is based on a rich stack of incremental algorithms to construct and modify concept hierarchies.

The *fuzzy lattice* structure in Fuzzy FCA depicts the notion of concepts that regroup the objects sharing the same attributes. In the same fashion, we use Fuzzy FCA to regroup cloud services according to their QoS offerings and users according to their similarity degrees. Also, using the fuzzy lattice's sub-hierarchy allows us to extract the most similar users and the positively rated cloud services. This allows reducing the time devoted to determine the top-k cloud services that cover the active user's query. One of the strengths of Fuzzy FCA is its ability to eliminate poorly rated services through the use of a threshold value. This allows having optimal clusters of cloud objects (e.g., candidate services, similar users, best rated services) and reducing the recommendation time.

5. Proposed approach

In this section, the main steps of our Fuzzy FCA based recommendation process are presented, as shown in Fig. 6. A brief description of each of the steps is also presented following the figure.

- ***Step 1:*** The active user submits a request of certain cloud services he likes to use. On a first level, the cloud's repository returns all of the available services that satisfy his requirements.
- ***Step 2:*** The Fuzzy FCA based recommendation process extracts all of the similar users to the active user. Two users share similar

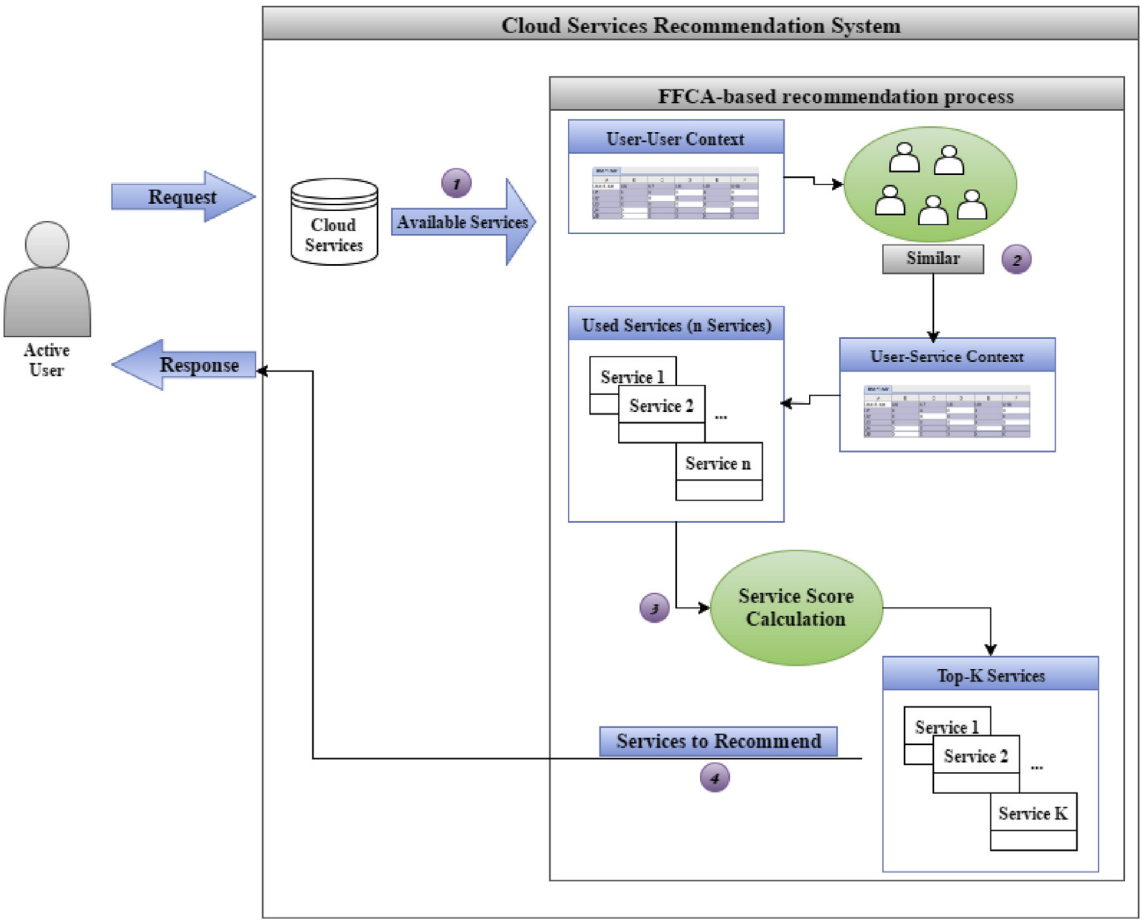

Fig. 6. Fuzzy FCA based recommendation process.

tastes if they have previously invoked a number of common cloud services.

- ***Step 3:*** The input of this step is the set of similar users. Our goal is to run through the history of the similar users and check if they used some of the candidate services. A score will then be calculated for each of those services.
- ***Step 4:*** After the score calculation, the final step is recommending the services having the highest score (top-k services).

To show how Fuzzy FCA is applied to the cloud service recommendation, we modelled the cloud environment and defined algorithms for cloud services evaluation. A brief description of those two main contributions is presented below:

- ***Cloud environment modelling using Fuzzy Formal Contexts and Lattices:*** This step consists of mainly creating three fuzzy formal contexts:
  – Cloud Services fuzzy formal context (K^{SC}) to model the quality of each cloud service.
  – Users Similarity fuzzy formal context (K^{UU}) to model the *active user's* neighborhood.
  – User-Service fuzzy formal context (K^{US}) to model the services used and evaluated by the similar cloud users.

Using the previously created fuzzy formal contexts as input, three fuzzy lattices will be created using one of the existing lattice construction algorithms [27]. The recommender system will extract, from those lattices, the cloud services that could be recommended to the targeted user.

- ***Fuzzy sub-lattices extraction algorithms:*** In this step, we will extract sub-lattices from the previously generated fuzzy lattices. A sub-lattice represents a smaller part of a whole lattice. It contains the information that we exactly need for the recommendation process. Such information might be candidate services, active user's neighborhood, similar users ratings of candidate services. An algorithm is needed to run through each fuzzy lattice for the extraction of the sub-lattice. A detailed description of the defined algorithms will be presented later on.

In the two following subsections, each of those two contributions will be described in detail.

Service-Criteria

A	B	C	D	E	F
Service-Criteria	Q1	Q2	Q3	Q4	Q5
S1	[0.4]	[0.6]	0	[0.5]	0
S2	[0.5]	0	[0.4]	0	[0.6]
S3	[0.2]	[0.4]	0	[0.7]	[0.5]
S4	0	[0.5]	[0.6]	[0.4]	0
S5	0	0	0	[0.8]	0
S6	[0.3]	[0.8]	[0.3]	[0.9]	[0.1]
S7	[0.7]	[0.5]	0	0	0

Fig. 7. Services fuzzy formal context (K^{SC}).

5.1. Cloud environment modelling using fuzzy FCA

In this section, the relations needed by our recommendation system will be presented using a set of fuzzy formal contexts. Then, a lattice will be generated out of each fuzzy formal context. Contexts and lattices were created using Galicia tool.[1]

5.1.1. Cloud services capabilities representation

To model the quality values offered by a cloud service, a *services fuzzy formal context* denoted K^{SC} is created, as shown in Fig. 7.

K^{SC} represents the available cloud services. Each service has a set of quality attributes denoted $Q = \{Q_1, Q_2, \ldots Q_n\}$. Example of quality attributes includes *availability*, *security*, *reliability*, *price*, *throughput, etc*. In our work, the set of quality attributes varies according to the used cloud services. For example, in case of IaaS services recommendation, quality attributes may include *VM resources capabilities, Storage offered, CPU, etc.* The quality values described in K^{SC} will tell if a service assures or has a high value of a certain criterion.

According to the K^{SC}, we will be able to define which of the services assures the highest criteria, so at the end of the recommendation process, the system will recommend the best of the available cloud services to the active user.

Fig. 7 shows that the service S_1 has an over average values of Q_2 and Q_4 (respectively 0.6 and 0.5), but a weak value of Q_1 (0.4). S_1 does not offer Q_3 and Q_5. On the other hand, S_6 offers great values of Q_2 and Q_4 (respectively 0.8 and 0.9) but comes with weak values for the rest of the criteria (values bellow $T = 0.5$). Empty entries in K^{SC} (e.g., $S_1.Q_3$) denote incomplete description of the cloud service.

Cloud services fuzzy lattice: To deal with the uncertainty and the lack of information on cloud services capabilities, a services fuzzy lattice denoted L^{SC} is created using *Bordat algorithm* [27] (see Fig. 8).

In the above fuzzy lattice, the Extent section represents the candidate services resulting from the retrieval phase and the Intent section represents each service's capabilities. The goal of this lattice is to only recommend services having the highest and the best QoS values.

For example, the concept C_{12} has the service S_6 in the *Extent* and all of the criteria in the *Intent* which implies that S_6 meets all of the criteria requested by the active user (with different quality values).

One of the strengths of Fuzzy FCA is to exclude useless data. In our case, all of the relations between services and their QoS attributes with quality values less than the specified threshold by the active user, are not shown in the lattice L^{SC}. This means that the selection and recommendation process will be limited to the cloud services that satisfy the active user's preferences. For example, in the concept C_{11} of Fig. 8, the quality attribute Q_3 is not found in the Intent section because the services of the Extent section do not offer that QoS attribute or the offered quality value of Q_3 is lower than the specified QoS level by the active user. This means that the score of this candidate service will be penalized during the selection phase because of the uncertain information regarding the quality.

In this version of work, we focus on the general aspect of the cloud service recommendation (recommend any type of service SaaS, IaaS or PaaS). In fact, our approach is generic enough to be applied in any cloud model or even in the case of emerging service models such as Business Process as a Service. Nevertheless, since each Cloud service model has its own specifications, and to offer a rich recommender system that covers all types of services, the cloud environment that includes a huge number of heterogeneous services can be modeled as a set of formal contexts, each one will be used to store the services of a specific model (e.g., SaaS formal context, IaaS formal context, BPaaS formal context, etc.). Hence, attributes in each formal context will vary depending on the cloud service type. The set of these formal contexts forms a complete repository of heterogeneous cloud services that are organized and grouped according to their properties, similarities and quality levels. Hence, unlike existing recommendation solutions which are designed to process services of a specific model, users in our approach can request various service models at the same time. In this case and for better processing of the user request, our Fuzzy FCA based recommender system will start by transforming the user request into a set of subqueries that will be processed against the involved formal contexts, according to the needed cloud service model.

[1] sourceforge.net/projects/galicia/.

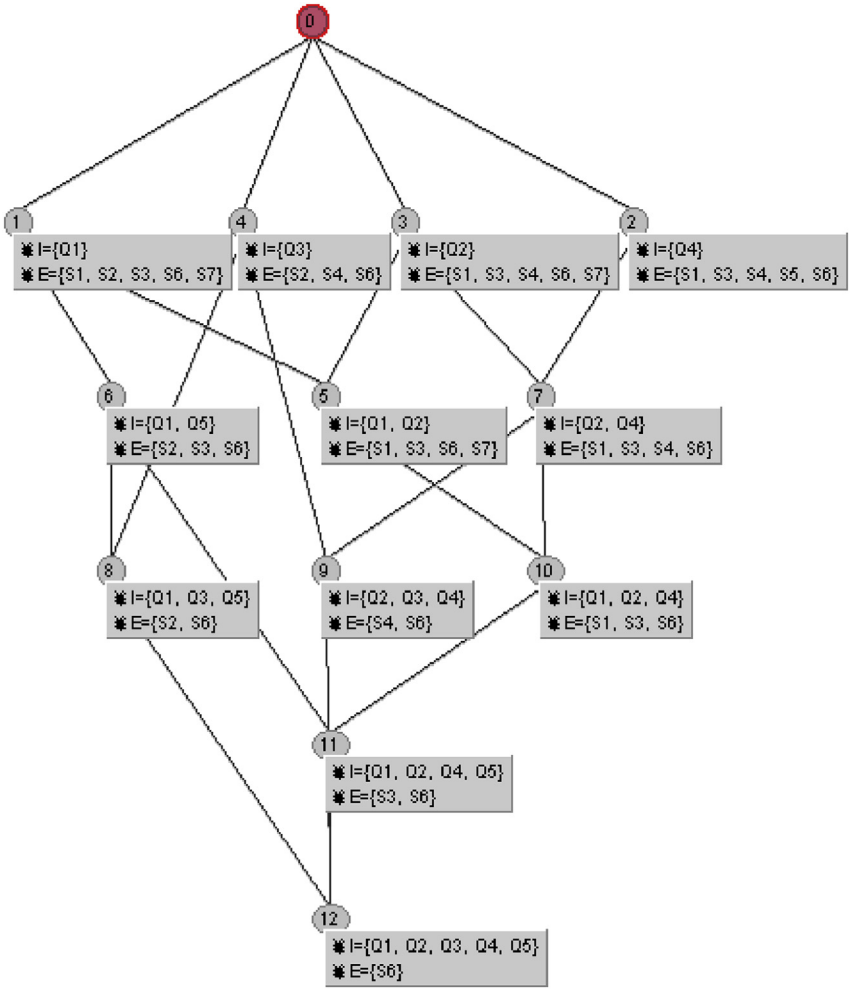

Fig. 8. Cloud services fuzzy lattice (L^{SC}).

5.1.2. Cloud users similarity representation

To model the users' neighborhoods, a *user-similarity fuzzy formal context* denoted K^{UU} is created. The objects represent users (the active user is included) and the attributes represent users too. The goal is to determine similar users to the targeted user according to the similarity degrees of the relation *user-user*.

K^{UU} is presented in Fig. 9.

Assuming that U_1 is the active user, looking at the users similarity fuzzy formal context, U_1 is similar to U_6, U_7 and U_9. According to the similarity degrees, he is most similar to U_7 (0.8) and less similar to U_9 (0.2). This means that U_7 shares the most consumed and rated services with U_1. From this information, the idea is that we can recommend to U_1 the cloud services that the most similar users to him used and liked. A threshold T will be defined later in the implementation. For example if $T = 0.5$, all the users that have the

User-User

A	B	C	D	E	F
User-User	U6	U7	U8	U9	U10
U1	[0.4]	[0.8]	0	[0.2]	0
U2	[0.6]	0	[0.5]	0	[0.7]
U3	[0.1]	[0.4]	0	[0.8]	0
U4	0	[0.3]	[0.5]	[0.7]	[0.3]
U5	0	[0.1]	[0.4]	[0.9]	[0.1]

Fig. 9. Users similarity fuzzy formal context (K^{UU}).

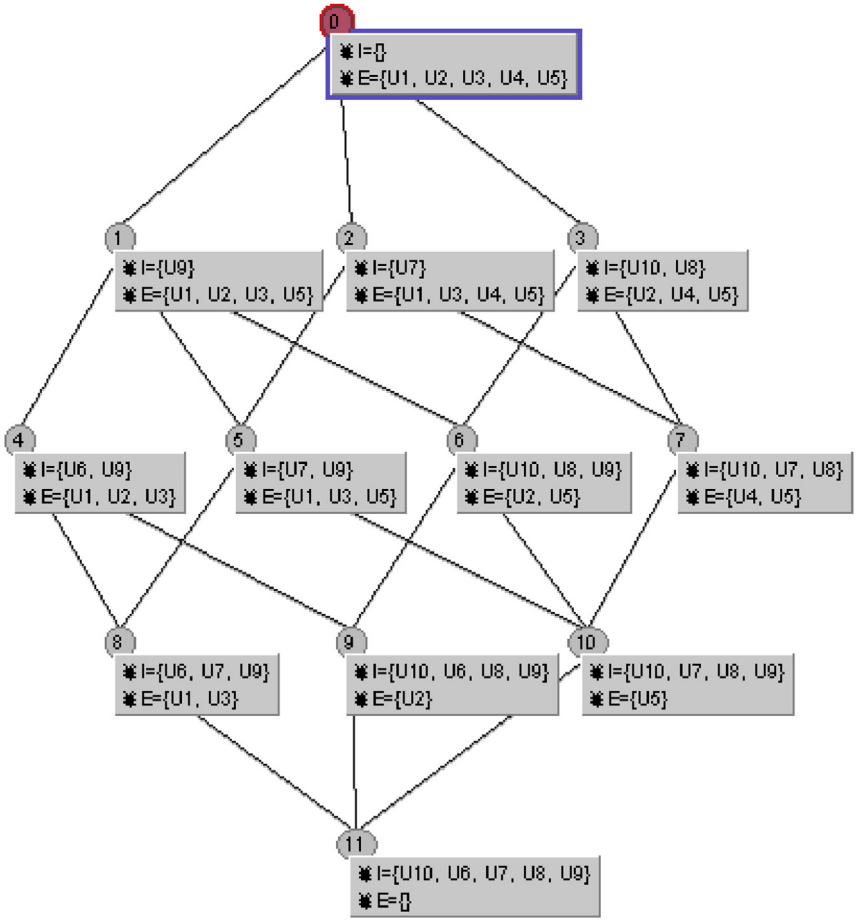

Fig. 10. Users similarity fuzzy lattice (L^{UU}).

degree of similarity below T will not be considered.

Users similarity fuzzy lattice: A users similarity fuzzy lattice denoted L^{UU} is created from the users similarity fuzzy formal context previously described, as shown in Fig. 10.

In L^{UU}, both of the objects and attributes represent the same thing, *users*. If a user U_i is found in another user's Intent section, it means that they are both similar (they have at least one service in common).

For example, using a *bottom-up* method (for optimization reason), we can find the active user's neighborhood in the second level of the fuzzy lattice, where the active user U_1 is in C_8.Extent. So, his neighbors are in C_8.Intent which are U_6, U_7 and U_9.

The goal behind this lattice representation is finding similar users to the active user. To extract such information, an algorithm is needed to run through all of the concepts. The relations between them, as well as the hidden relations, will also be addressed in the next section.

5.1.3. Cloud services ratings representation

To model the degree of user satisfaction after consuming a cloud service, a *services ratings fuzzy formal context* denoted K^{US} is created, as shown in Fig. 11.

K^{US} is extracted according to the similar users only. Lets say that we have m selected services in our recommendation process. The

User-Service

A	B	C	D	E	F	G	H
User-Service	S1	S2	S3	S4	S5	S6	S7
U6	[0.4]	[0.6]	0	[0.2]	0	[0.9]	0
U7	0	[0.1]	[0.4]	0	[0.8]	[0.3]	0
U9	[0.8]	0	[0.1]	0	0	0	[0.7]

Fig. 11. Services ratings fuzzy formal context (K^{US}).

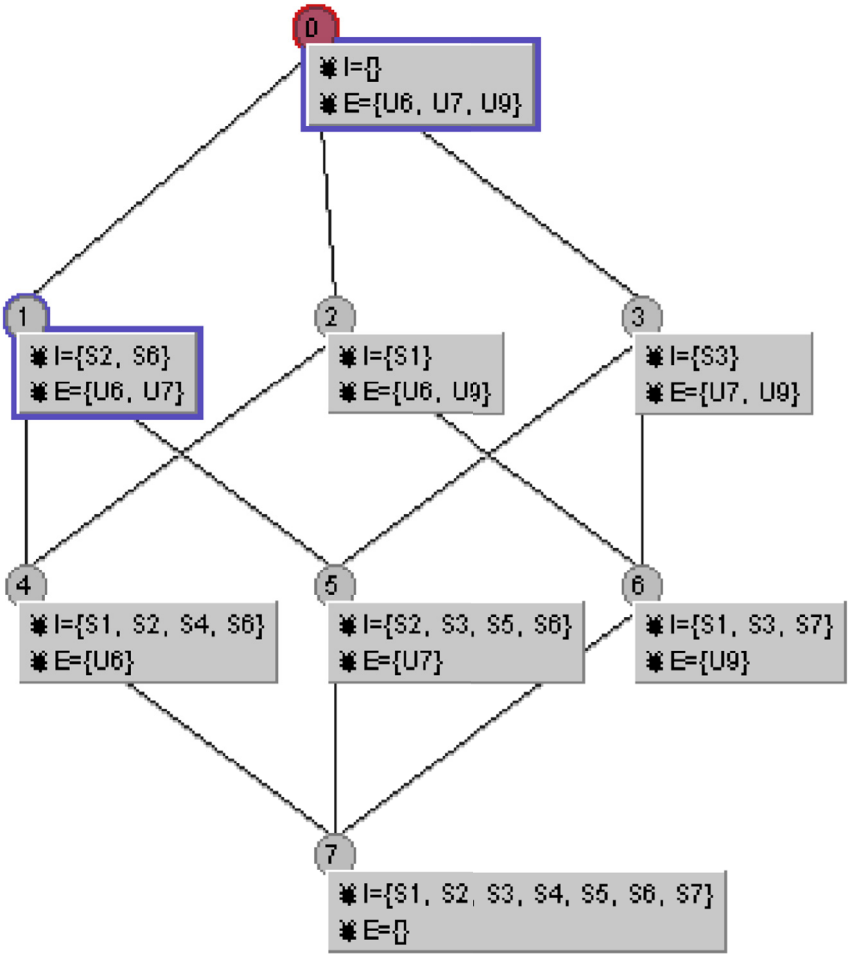

Fig. 12. Services ratings fuzzy lattice (L^{US}).

rows represent the neighbors (or similar users) of U_a. The columns represent a set S of m cloud services $S = \{s_1, s_2, \ldots, s_m\}$ used by the neighbors of the active user. In the following example, U_1 is assumed to be the active user.

K^{US} presents the history of each neighbor of the *active user* U_1. According to it, we will determine cloud services that are liked by the similar users. Once the services are extracted, we can recommend them to the active user since his neighbors usually have similar interests and have consumed similar services in common. So, the active user may like the services positively rated by his neighbors too.

For example, Fig. 11 shows that U_6 liked the services S_2 and S_6, as he positively rated them (respectively 0.6 and 0.9). But he poorly rated S_1 and S_4 (respectively 0.4 and 0.2). He did not use the services S_3 and S_5.

U_7 really liked S_5 (0.8) but poorly rated S_2, S_3 and S_6 (respectively 0.1, 0.4 and 0.3), and did not use S_1 and S_4.

Services ratings fuzzy lattice To exclude the poorly rated cloud services from the candidate services set, a Services Ratings fuzzy lattice denoted L^{US} is created, as shown in Fig. 12.

The Extent here contains the similar users, extracted from the previous fuzzy lattice (L^{UU}). The Intent contains the services used and positively rated by them. Indeed, the fuzzy extension of FCA allowed us to ignore the services with a rating value that is below the given threshold by the active user. The goal behind this lattice representation is extracting the neighbors history which is all of the cloud services that they used and liked. Using that information, we will be able to have an idea of what the active user might like.

For example, using a bottom-up method, according to the concepts of the second level (C_4, C_5 and C_6) the user U_6 liked the services $\{S_1, S_2, S_4, S_6\}$, U_7 liked $\{S_2, S_3, S_5, S_6\}$ and U_9 liked $\{S_1, S_3, S_7\}$. From such information, we can conclude that all of the services $\{S_1, S_2, \ldots, S_7\}$ can be recommended to the active user U_1 since his neighbors used them. However, those services need to be filtrated and only the ones with the highest scores (Top_k) will be recommended, as we will show in the next section.

5.2. Recommendation process

In this section, we present the recommendation process. Three extraction algorithms are defined to run through the fuzzy lattices, previously created (L^{SC}, L^{UU}, and L^{US}), and to pull out the pertinent information for our recommendation process. A general overview

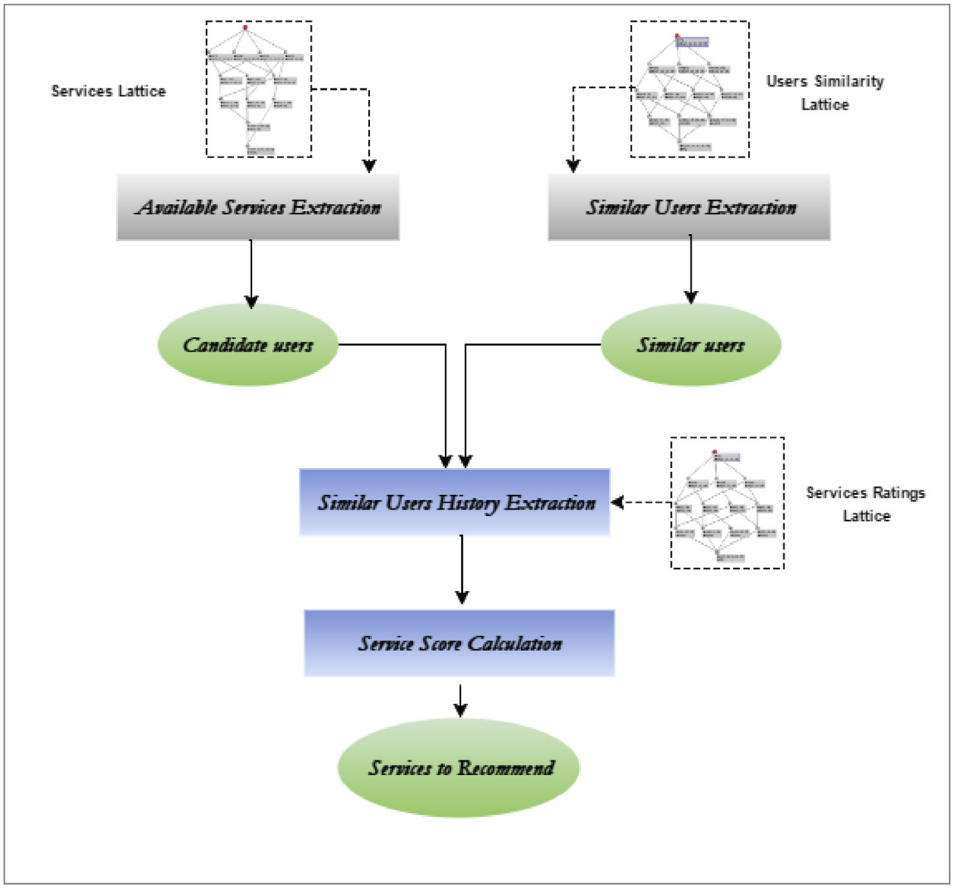

Fig. 13. Recommendation steps.

of the recommendation steps is presented in Fig. 13.

As shown in Fig. 13, the available services' extraction step is based on the service-criteria fuzzy Lattice. However, the similar users' extraction step takes the users similarity fuzzy Lattice as input. The extracted sub-lattices (pertinent information) from those two steps will be the input of the next step, which is extracting the similar users' ratings of their previously used services. The output of this step is a fuzzy sub-hierarchy of concepts containing the services used by the similar users of U_a, among the available services. A score will, then, be given to each of those services. In the end, the services having the highest scores will be recommended to U_a.

Next sub-sections will give more details on each of the steps above.

5.2.1. Candidate services sub-lattice extraction

Once the active user submits his request, the first step of our system, before any recommendations, is to extract all of the candidate cloud services that satisfy his requirements. That information will be extracted from the *Cloud Services Fuzzy Lattice* (L^{SC}) using Algorithm 1.

The algorithm takes as input the active user's (U_a) request (a set of requested services *RS*) and the lattice L^{SC} that contains all of the cloud services. As output, it returns a set of available services *AS* that satisfy U_a's preferences.

The goal of this algorithm is to define all of the cloud services that meet with U_a's request. It runs through every concept of the cloud services lattice L^{SC} looking for the requested services *(lines 2–9)*. For each service S_j, we test if it can be found in the concept's Extent part by running through *RS (lines 3–8)*. If a requested service is found in a concept's Extent section, a score will be calculated for that service to see if it satisfies the quality criteria, that is, the active user's threshold *T (line 4)*.

The Score function is described as follows:

$$\mathbf{Score}(S_j) \leftarrow \left(\frac{\sum_{i=1}^{n} (W_i.\ Cr_{ij}).\ P}{n} \right)$$

where n is the number of criteria defined for the jth service, Cr_{ij} is the ith criterion of the service S_j. Each criterion Cr_{ij} has a weight W_i to indicate the importance degree of the criterion for the active user. We choose to add a penalty P in the score calculation. P takes the value 0 when one of the criterion does not satisfy the threshold T or is missing. That will ensure a pertinent recommendations later on. If the conditions stated above are met, the service will be saved in the available services' set *AS (line 5)*.In the end, we will have a

set of the services that can be recommended later in the recommendation process *(line 10)*. This set will be the input of the service ratings sub-lattice's extraction step (see section 5.2.3).

In the next subsection, a detailed description of the similar users extraction algorithm is presented as it is the next step of our recommendation process.

Algorithm 1
Available Cloud Services Sub-Lattice Extraction.

Input : *Lattice of Cloud Services* L^{SC}, *Set of Requested Services* RS
Output: *Set of Selected Services* $AS = \{S_1, S_2, .., S_n\}$.

```
1  AS ← ∅;
2  foreach Concept C_i in L^SC do
3      foreach Service S_j in RS do
4          if (S_j ∈ C_i.Extent) and ( Score(C_i.Intent) ≥ T ) then
5              AS ← AS ∪ S_j
6              RS ← RS − S_j
7          end
8      end
9  end
10 Return AS;
```

5.2.2. Similar users sub-lattice extraction

After extracting the services that satisfy the active user's requirements, the idea is to get all of his neighbors, since similar users tend to have alike tastes and interests. This allows to filtrate the available services according to each of the neighbor's history (used and positively rated services). So in the end, only the best services will be recommended.

The algorithm below takes U_a and the users similarity lattice L^{UU} as input. As a result, it returns a set of similar users to U_a.

Based on L^{UU}, if the active user is to be found in the *Extent* part of any concept, this means that he is similar to all of the users in the *Intent* part. The algorithm runs through every concept of the similarity fuzzy lattice using a *bottom-Up* method for optimization purposes (*lines* 2–7). The goal is to look for similar users to U_a. For each concept in L^{UU}, we test if U_a is found in a concept's Extent section (*lines* 3–6). When he is found, the users in that concept's Intent section will be saved in the similar users' set *SU* (*line 4*). This later will be the second input of the next step (users history sub-lattice extraction).

Algorithm 2
Similar Users Sub-Lattice Extraction.

Input : *Active user* U_a, *Users Similarity lattice* L^{UU}
Output: *Set of Similar Users* $SU = \{U_1, U_2, .., U_n\}$.

```
1  SU ← ∅;
2  foreach Concept C_i in L^UU do
3      if U_a ∈ C_i.Extent then
4          SU ← SU ∪ C_i.Intent
5          Break;
6      end
7  end
8  Return SU;
```

5.2.3. Services ratings sub-lattice extraction

In this section, the outputs of the previous two steps (available services set *AS* and similar users set *SU*) are used to determine the top-rated Cloud services.

With the available services set, we know all of the cloud services that might satisfy the active user's requirements. On the other hand, the similar users' set will be used to extract a ratings sub-lattice that has only the similar users and their ratings. Having those two sets in hand, we will be able to choose and only recommend the services that were previously used and positively rated by the similar users.

For that purpose, the following algorithm is presented. It takes the available services set (*AS*), the similar users set (*SU*) and the services ratings lattice (L^{US}) as input, and returns a set of services (*RS*) with the highest scores that will be recommended to the active user.

Algorithm 3
Ratings Sub-Lattice Extraction.

Input : *AS (Set of Available services)*, L^{US} *(Ratings Lattice)*
Output: *RS (Set of* Top_k *Services to Recommend)*

1 $RS \leftarrow \emptyset$;
2 *Initialize ServiceRatings (Table of Available Services Ratings)*;
3 **foreach** *Service* S_j *in AS* **do**
4 **foreach** *Concept* C_i *in* L^{US} **do**
5 **if** $S_j \in C_i.Intent$ **then**
6 $m \leftarrow |C_i.Extent|$
7 $R \leftarrow 0$
8 **foreach** *User* u_k *in* $C_i.Extent$ **do**
9 $R \leftarrow R + (\frac{Rating(u_k,S_j)m}{n}).\boldsymbol{sim}(U_a, u_k)$
10 **end**
11 *ServiceRatings*$[j] \leftarrow R$
12 $RS \leftarrow RS \cup \{S_j\}$
13 $AS \leftarrow AS - S_j$
14 **break;**
15 **end**
16 **end**
17 **end**
18 ***Sort*** *ServiceRatings[] in a Descending order.*
19 **Return** $RS.Top_k()$;

First, Algorithm 3 runs through the candidate services set *AS (lines 3–17)*, to compute a global rating for each service S_j. To find how many times a candidate service was used and rated by similar users, a run through the ratings lattice L^{US} is needed *(lines 4–16)*. If a candidate service is to be found in the Intent section of a concept C_i, we run through every user of the Extent section to compute a rating for that service *(lines 5–10)*. The computed rating value will be saved in *ServiceRating* table and the service S_j will be added to the recommendation set *RS* since it was used and positively rated by U_a's neighborhood *(lines 11–12)*. The service S_j will, then, be removed from the set *AS* of available services *(line 13)*. Finally, the ratings saved in the *ServiceRatings* table will be sorted in a descending order and only the cloud services with the highest scores (top_k) will be recommended *(lines 18–19)*.

To grant a score for each of the services consumed by the similar users, we defined a function that allows to predict wether the service in question will be liked or not by the active user.

Let n be the number of similar users to U_a, m the number of users that consumed the service S_j, and $Rating(u_i, S_j)$ the rating given by the user u_i for the service S_j. The degree of similarity between the active user and his neighbors (a value in [0, 1]) is denoted as $sim(U_a, u_i)$.

The function is defined as follows:

$$f(S_j) = \left|\frac{Rating(u_i, S_j)m}{n}\right|. sim(U_a, u_i)$$

6. Theoretical complexity study

Fuzzy FCA assures high quality results because of its strong mathematical background and the effectiveness of the lattice representation. But such thing must come with a cost. The complexity of fuzzy FCA can be high sometimes, depending on the number of concepts in a lattice.

To establish the all-round complexity of our recommendation process, a complexity study was conducted on all of our proposed algorithms. The first algorithm represents the selection process of candidate services. The second algorithm defines the similarity (similar users to the active user) and the last algorithm represents the final step, which is the recommendation of top_k services.

Each algorithm's complexity was studied separately as detailed bellow:

- **Algorithm 1:** In this algorithm, the complexity depends on the size of the services lattice L^{SC} and the set of requested services *RS*. In the worst case, we run through the whole services lattice along running through the requested services set (*lines 2–9* in algorithm 1). So, a total of $|L^{SC}|$ concepts will be processed with each concept being ran through $|RS|$ times. Which gives us a total complexity in the order of $O(|L^{SC}|.|RS|)$.
- **Algorithm 2:** The complexity of this algorithm depends mainly on the number of concepts in the similarity lattice. In the worst case scenario, the active user may belong to the Extent part of the Infimum (bottom concept of the lattice). So, we will have to run

Table 2
IaaS user request.

Attribute	Value	Weight
CPU	1.6 GHz	
Memory	1.5 GB	
Storage	500 GB	
Accuracy	98.5%	0.4
Failure rate	0.01	0.1
Stable time	1 h	0.3
Throughput	100 Mbps	0.2
VM response	85 ms	

through the whole similarity lattice to find the similar users (*lines 2–7* in algorithm 2). This gives a total complexity of $O(|L^{UU}|)$.

- **Algorithm 3:** The complexity of the third algorithm is based on three inputs: the number of candidate services $|AS|$, the size of the ratings lattice $|L^{US}|$ and the number of similar users to U_a $|SU|$. For each cloud service, the algorithm processes all the concepts in L^{US} (*lines 3 and 4* in algorithm 3), which gives a complexity in the order of $O(|L^{US}|.|AS|)$. Since the evaluation of each candidate service requires checking all the services in the Intent part (*lines 5–15*) and all the users in the Extent part (*lines 8–10*), this gives a complexity in the order of $O(|AS|.|SU|)$ in the worst case. Hence, the complexity of this algorithm would be in the order of $O(|AS|^2.|SU|.|L^{US}|)$.

7. Recommendation scenario

To demonstrate the effectiveness of our novel approach, we proposed a cloud recommendation implementation that illustrates the IaaS cloud model. Eight attributes of the IaaS services were taken into account throughout the recommendation process: CPU capacity, Memory, Storage, Accuracy, Failure rate, Stable time, Throughput and VM response.

In the implementation step, we considered as example scenario, a simple application that should be deployed on a Cloud zone. The selection of the most performant placement is a challenging task, due to the huge number of IaaS providers (i.e. size of the services' formal context) who may satisfy the user request and can provision the hosting VMs; and also due to the various satisfaction degrees of cloud consumers (i.e. size of the ratings' formal context). For this, the goal is to find the most optimal cloud availability zone that offers multiple service instances (e.g., VM instances) with a high performance level. The user's request (resource requirements and their importance degrees) is depicted in Table 2.

The selection and recommendation are based on the criteria specified in the user request. In our IaaS recommendation scenario, we divided a user request into two main parts: service requirements and preferred execution context. The first part includes resource requirements (e.g., storage, CPU, memory, etc.) and QoS requirements (e.g., required latency and performance levels), and will be used to recommend the best IaaS services. Whereas the second part describes the required execution environment in terms of service provisioning constraints and policies, preferred machines types, etc. Such requirements will be useful to refine the recommendation results by selecting the optimal availability zone. In the current implementation, we only consider the first part of the user request.

8. Experiments and evaluation

An implementation for the cloud service recommendation process is proposed based on the above IaaS recommendation scenario. The prototype was developed using Java as programming language and Netbeans as IDE. For the fuzzy lattices construction and the pertinent information search process, we used *Galicia* library and an existing Fuzzy FCA implementation.

Implementation details, as well as the used datasets, are available on the following URL: goo.gl/Df5WyP.

For all test problems, we generated the cloud environment objects (services data, users neighborhood, users ratings) as a set of fuzzy formal contexts. These latter are created from the random data of 500 cloud services and 1000 users, and represent the data sets used in the experimental studies to evaluate our approach. The ratings fuzzy formal context contains up to 400000 user-service interactions when its density reaches 80%.

Some entries in the formal contexts were randomly removed to stimulate different levels of data sparsity.

The set of experiments was conducted on a Hp Pavilion dv6 laptop with Intel core i7-2630QM processor, 4Go of Ram and 750Go Hitachi hard drive. The first set was focused on the computation time of the recommendations, whereas the second set focused on the quality of the generated recommendations.

Each experiment will be detailed and discussed in the following subsections.

8.1. Computation time experiments

Despite the fact that Fuzzy FCA is known to generate quality results, it is also known to struggle with the computation time. In fact, running through the lattice can sometimes be time consuming, specially when the data density is high. In this section, we study the performance of our approach in terms of computation time against increasing the complexity of the cloud environment (i.e. users, services, ratings, etc.).

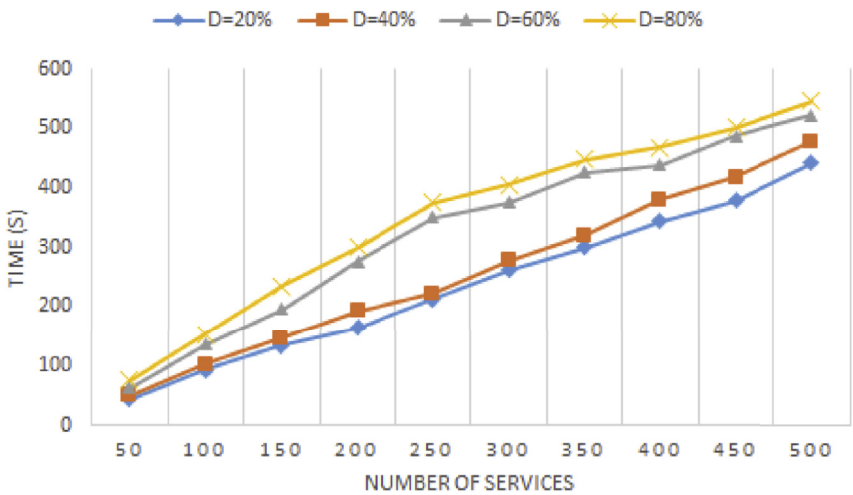

Fig. 14. Recommendation time under various numbers of cloud services and context densities.

8.1.1. Recommendation time with various numbers of services

The first set of experiments involves varying the number of available cloud services between 50 and 500, while fixing the number of users to 500. Each test was executed with different data densities (20%–80% with a step of 20%) of the fuzzy formal contexts. The behavior of our recommender system, in terms of total execution time, is presented in Fig. 14.

It is evident that the computation time increases linearly as the number of services and the context density values climb up. Fig. 14 shows that we have an execution time less than 102s when the number of services is less than 100 and the data density is below 40%. The execution time increases significantly when the number of services surpasses 250 and the density of the services formal context is over 60%, because running through a fuzzy lattice becomes more complex when the numbers of its objects (services) and attributes (criteria) increase. The execution time is over 340s when the number of services is up to 400, for all density values. However, we notice an improvement in the increase rate of the recommendation time when the formal contexts become larger. For example, the increase in recommendation time is between 169s and 253s for a 100-to-300 services variation, while the increase rate did not go beyond 84s (for 60% and 80% density) and 98s (for 20% and 40% density) when we varied the services number between 400 and 500. This is explained by the adopted bottom-up approach that limits the search process to the first few levels of the lattice (i.e. minimal number of formal concepts), even with a large number of Cloud services, without needing to run through the whole lattice. In contrast, a low density implies a sparse formal context and, consequently, a higher number of small-size formal concepts. This means that the recommendation algorithm will find the similar users and the top-rated services in a group of Intents, unlike the denser formal context that allows generating clusters (formal concepts) with a larger size. Hence, we can find the similar users and the requested Cloud services in few formal concepts, or a single one in the best case. The impact of the formal context density on the gap between recommendation times can be clearly seen when the number of cloud services reaches 500. But in the end, the recommendation time is always considerably high and should be reduced.

Parallel processing models such as MapReduce [30] may be good candidates to this problem. MapReduce has been successfully combined with FCA, in the work of Ferchichi and Akaichi [53], to search for the k-nearest neighbors in complex road networks. By combining Fuzzy FCA and parallel programming, the complexity of the recommendation process can be considerably reduced. This is by partitioning the ratings formal context into smaller ones, that will be processed in parallel fashion, which has a positive impact on the recommendation time. So, Fuzzy FCA can be accepted as an effective solution even in case of a large formal context of user-service ratings.

8.1.2. Recommendation time with various numbers of users

These experiments involve varying the number of users (100..1000) and the values of the generated top_k services (5..20) for different values of data density (20%.. 80%), each time. The number of cloud services is fixed to 250. Results are presented in Fig. 15.

Looking at Fig. 15, the execution time is below 96.47s when the number of users is less than 100. The computation time becomes more significant as the number of users increases along the ratings densities as well, which is expected. The more the ratings density is high, the more complex and time-consuming running through the ratings lattice becomes. We also observe that when the top_k values increase, the time needed to generate those top_k recommendations increases as well. In fact, extracting more top_k services demands running through the service ratings fuzzy lattice to process additional formal concepts, which is time consuming.

We could also notice that our recommendation method features the same performance behaviour as in Fig. 14. Indeed, the increase rate in the recommendation time was improved when the number of cloud users becomes high. Taking the example of 20% density and top_{10} case, the recommendation time has increased by 303.47s for a 100-to-700 users variation, while the increase rate did not exceed 176.2s when we varied the number of users between 800 and 1000. Again, the adopted bottom-up parsing method allows finding the similar users in the first few levels of the similarity lattice. These levels contain the concepts with the largest

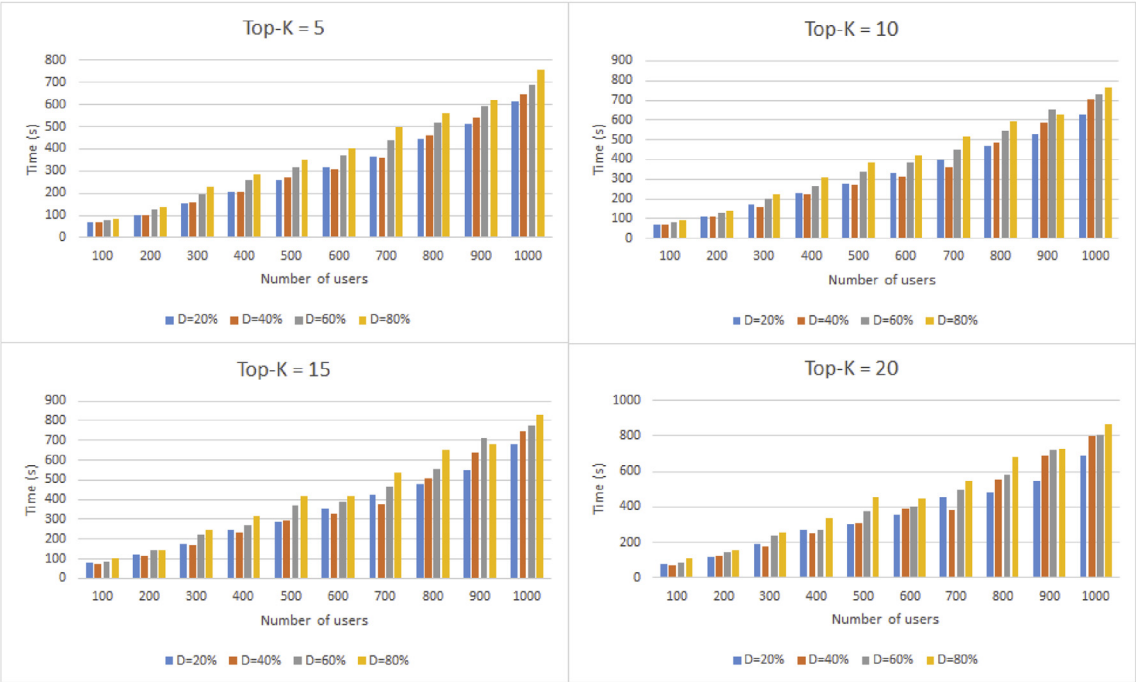

Fig. 15. Recommendation time with various numbers of users under different Top_k values.

Extents (largest subsets of Cloud services and similar users) in the lattices. This also means that our recommendation method performs better with large ratings matrices.

It is important to mention that, in real-world scenarios, there exists a large amount of (big) data including the high number of clouds, providers, various types of services (e.g., conversational and transactional services, data services, and recently big services) that produce a huge volume of data and users' ratings. So in real-life application, this will generate a large formal context with overlapping relationships. In specific scenarios, FCA solutions may risk to suffer from a high complexity and execution time, for example, when a system needs to repetitively build the lattice from the formal context. This is the case of dynamic or real-time applications or stream-based recommender systems, that need to rebuild the ratings lattice to make a decision or to process the new coming/generated information (new users, new services, new ratings, etc.). In the future, we will adapt our approach in the context of stream-based recommendation, to take the highly dynamic nature of the cloud environment into account (e.g., streams of users' ratings), which causes frequent updates in the formal contexts. Incremental algorithms for lattice construction may be useful in this case.

8.2. Recommendations quality experiments

The purpose of this set of tests is to see if by decreasing the density of the ratings' formal context, the quality of recommendations will be affected. To analyze that aspect, a widely used evaluation metric called Mean Absolute Error (MAE) [32] was exploited to evaluate the quality of the recommendations generated by our approach, in comparison with other approaches. MAE is defined as follows:

$$MAE = \left(\frac{\sum_{i,j} \left| F_{ij} - Y_{ij} \right|}{N} \right)$$

where N presents the total number of criteria. F_{ij} and Y_{ij} are, respectively, the actual and predicted rating values of the ith user for the jth service.

MAE will be calculated with each density variation to evaluate the quality (i.e. score) of the recommended services. MAE and quality values are in [0, 1].

This test was carried out by varying the density of the ratings formal context between 10% and 50%, while fixing the number of cloud services and users to 250 and 500 respectively. Top_k value is set to 10.

The results in Fig. 16 show that with increasing the density of the ratings formal context, the MAE value is gradually reduced. MAE is less than 0.15 in most cases and below 0.11 at the end. Results show as well that, even with low ratings density, the average quality of the top-10 recommended services is high (0.81 when the ratings formal context density is at 10%). It's the strength of Fuzzy

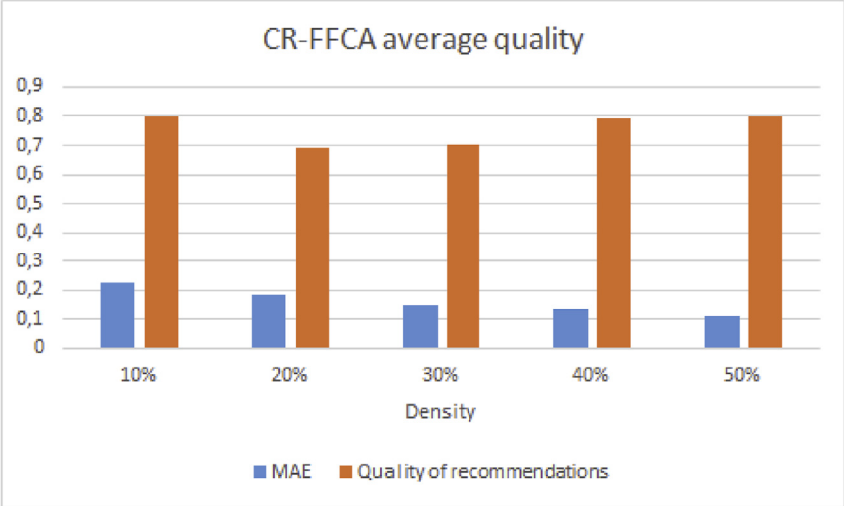

Fig. 16. Recommendations quality analysis.

FCA. In fact, as described in section 5.1.3, the unrated or poorly rated services will not be considered during the recommendation process, which increased the accuracy of the recommendations. That explains why the average quality value is high whether the ratings density is low or high.

The results show as well that the quality of the recommendations will not get below the value 0.6. That's another huge advantage of Fuzzy FCA as it allows setting a threshold for service ratings. Any service having a score value below the specified threshold will be excluded from the ratings fuzzy lattice, which explains the high quality of the recommendation results (0.69 in the worst case and 0.8 in most cases) produced by our Fuzzy FCA based approach.

It is clear that the improvement in the recommendations' quality is obtained by a higher density of the ratings fuzzy context. This implied a larger number of user-service interactions and consequently a better view and evaluation of cloud services quality. This is unlike the low density (case of 10% and 20%), which limited the recommendation to a few number of entries (i.e. users ratings) and, consequently, has affected the quality of recommended services (MAE values between 0.19 and 0.22). But in the end, our approach produces good and accurate recommendations compared to existing solutions, as we will show in Section 8.3.1.

8.3. Comparison with different approaches

In this section, we compare our approach (denoted CR-FFCA) to *TSR-INF* [54], *WSRec* [33,34] and *Time-aware WSRec* [35], in terms of MAE values and recommendation times.

8.3.1. Recommendation quality comparison

Four tests were carried out to study the accuracy of the compared approaches. These tests involve varying data density (10%..40%) for different values of top_k services (2..20), while the number of cloud users and services is fixed to 500 and 250 respectively. Results are shown in Fig. 17.

According to the results, CR-FFCA provides the best quality of recommendations compared to the other three approaches. The MAE values are low in comparison to WSRec, TA-WSRec and TSR-INF (0.079 at the end). WSRec's MAE value is always high (between 0.307 and 0.399 in the best case). MAE values produced by WSRec-TA become close to those of CR-FFCA, in case of higher densities (30% and 40%). But, CR-FFCA produces the most accurate results with an error rate equal to 0.216, in the worst case (0.327 for WSRec-TA). Results show as well that even with low data densities, our MAE values are better than the other approaches. In fact, the unrated or poorly rated services, as well as the less similar users will not be considered from the start in our recommendation process, which means that our recommendation system will process only the fuzzy formal concepts that contain the services with the highest quality and the most similar users. On the other hand, the denser the formal contexts get, the better and more accurate our recommendations will be (MAE = 0.216 when the density = 10%, while it is below 0.087 when the data density is at 40%).

Regarding the quality of recommendations produced by TSR-INF, it is clear that MAE values steadily increase with the decreasing of the density of QoE evaluation matrix, and reached 0.367 when the density is less than 20%. In the best cases, MAE values are equal to 0.181 and 0.175 when the density reaches 30% and 40%, respectively. The recommendation accuracy of TSR-INF approach is better than WSRec, but it is high compared to the accuracy of CR-FFCA. Also, it can be seen from TSR-INF results that the gap of error rate enlarges with higher density (case of 30% and 40%). Although TSR-INF effectively expresses the uncertainty of service trustworthiness using the eigenvalue interval [54] and takes advantage of the user-side feature information, the recommendation accuracy is influenced by the adopted trustworthiness measurement method and the impact of parameters for interval number.

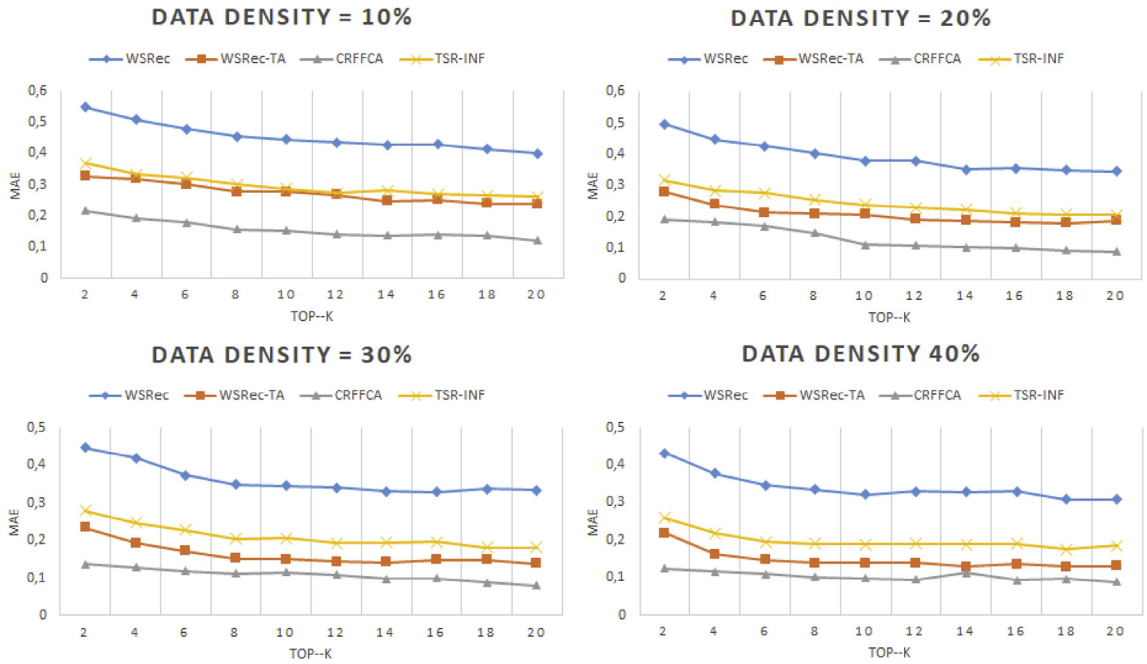

Fig. 17. Recommendation quality evaluation under different data densities.

8.3.2. Recommendation time comparison

In these tests, we varied the size of the ratings formal context in terms of number of Cloud users (between 100 and 1000) and ratings densities (between 10% and 40%). The number of cloud services is fixed to 250. The average recommendation times for the different approaches are shown in Fig. 18.

Generally speaking, the computation time of CR-FFCA is slower than those of the three recommendation methods. TSR-INF is fastest, WSRec and WSRec-TA are the next and CR-FFCA is the lowest. For all density values, the recommendation time of CR-FFCA increases proportionally to the number of users. It is evident that the gap between computation times increases in case of higher densities and larger ratings formal context. The recommendation time is considerably affected by the size of lattices and the

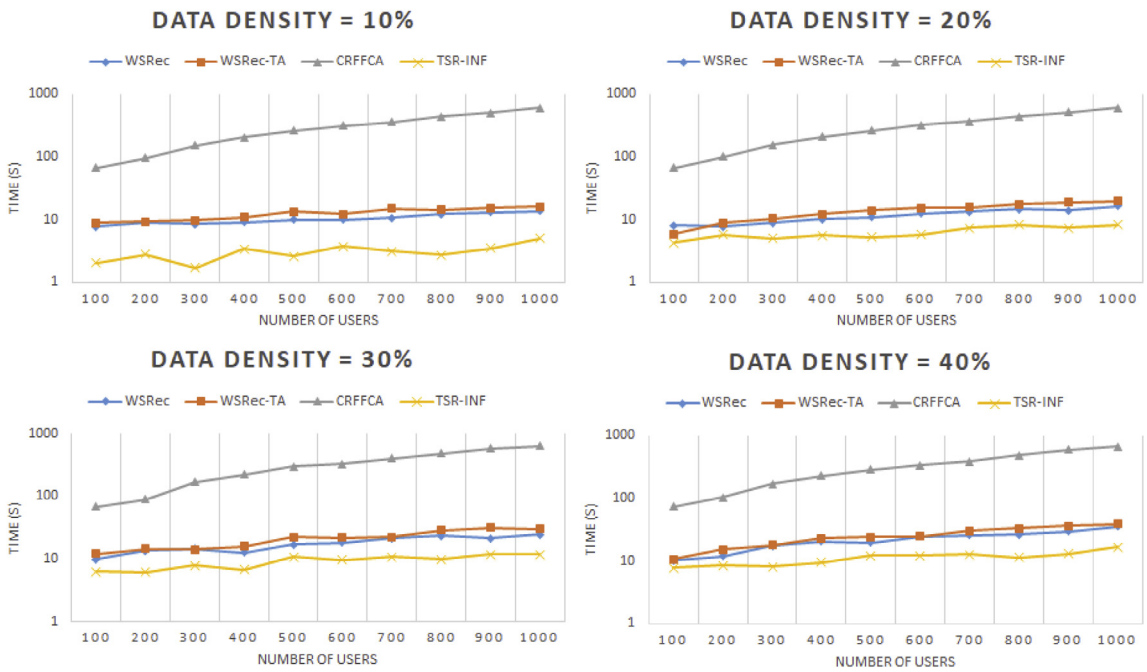

Fig. 18. Average computation times with different densities and numbers of users.

overlapping relations between formal concepts. In fact, the complexity of the ratings lattice implies more iterations to check the membership of each candidate service to the set of formal concepts, and to process similar users in order to calculate the score of each candidate service.

All these experiments indicate that CR-FFCA produced accurate results on the whole. However, it requires a considerable processing time when the complexity of the ratings formal context increases. This will be part of the future improvements.

9. Conclusion and future work

Cloud computing is gaining much popularity nowadays, and the amount of offered cloud services is increasing exponentially. To cope with the challenging task of optimal service selection, we proposed a cloud service recommendation system that helps cloud users finding exactly what they are looking for in a service. Fuzzy Formal Concept Analysis was adopted to generate reliable recommendations using the lattice representation. This method allowed to transform the cloud service repository into a set of small clusters, where the relations between high quality services and users with best experiences are organized using fuzzy formal concepts. The fuzzy lattice representation allowed to exclude useless cloud related data. This is by hiding poorly rated cloud services and eliminating the less similar users which share few mutual services with the active user. That limited the recommendation process to the best rated services and the most similar users, guaranteeing hence accurate recommendations.

In our work, the explicit hierarchical representation of Cloud information (users, services, ratings, etc.) has allowed a better grouping and analysis of such information. This is unlike most existing approaches that model cloud services data as a matrix of users' ratings with vector structures devoted to store QoS information, and without focusing on the relations between users.The experiments have shown that our approach allows an efficient service selection and recommendation for the targeted user. But it still can be enhanced on different levels. We intend to conduct more experimental studies, in order to evaluate the effectiveness and accuracy of the fuzzy FCA based approach.

The mathematical foundation of Fuzzy FCA method has ensured a high level of accuracy and trust in the recommended cloud services. However, the time spent to recommend the suitable cloud services adds more complexity to the Fuzzy FCA-based recommendation approach, as the complexity of parsing large concept lattices depends on the number of cloud services and the degree of changes in their properties and ratings. The future work aims to resolve the major issues related to exploring the cloud environment, as well as reduce the time spent to deliver a suitable cloud service to the active user. This will be achieved by combining the strengths of Fuzzy FCA, as a powerful mean of clustering the cloud services related information, and the processing capabilities of MapReduce, as a well-known parallel programming model [30]. MapReduce framework will be used to effectively deal with the highly distributed selection and recommendation process across a huge number of cloud services and Cloud availability zones. To meet this goal, we intend to model each cloud zone using the three formal contexts discussed in the previous section, then, parallelize the recommendation process across a set of mapper and reducer machines; each one takes as input a set of cloud formal contexts according to the availability zone.

Also, most existing models on Cloud service recommendation are static, while in the real world, the perception and popularity of Cloud services may continually change. Time is becoming an increasingly important factor in recommender systems since time effects influence users' preferences to a large extent. To make our proposed recommender system time-aware, we will combine a temporal extension of Fuzzy FCA and a well-known prediction algorithm namely Slope One [20]. FCA has been extended with time dimension, in some works, in order to facilitate the temporal data analysis and knowledge discovery [56,57]. In our future work, the temporal extension of Fuzzy FCA will be exploited in order to reorganize users into a set of fuzzy concepts, where each one represents similar users. The users' clustering will be based mainly on both classic similarity (i.e. commonly consumed services) and temporal similarity. This later could be computed using time parameters like the use of the recent dates of invocation, the succession of events, the distance between timeframes, etc. This step will return as output a set of time-aware fuzzy concepts containing users with similar timeframes and commonly consumed services. As for the time-based Slope One method, this step will be useful for predicting the missing ratings of Cloud services that have not been invoked by users over time.

References

[1] P. Mell, T. Grance, The NIST Definition of Cloud Computing, National Institute of Standards and Technology, 2011.
[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, ... M. Zaharia, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.
[3] S.M. Han, M.M. Hassan, C.W. Yoon, E.N. Huh, Efficient service recommendation system for cloud computing market, Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, ACM, 2009, November, pp. 839–845.
[4] Z. Gui, C. Yang, J. Xia, Q. Huang, K. Liu, Z. Li, ... B. Jin, A service brokering and recommendation mechanism for better selecting cloud services, PLoS One 9 (8) (2014) e105297.
[5] S. Senatore, G. Pasi, Lattice navigation for collaborative filtering by means of (fuzzy) formal concept analysis, Proceedings of the 28th Annual ACM Symposium on Applied Computing, ACM, 2013, March, pp. 920–926.
[6] L. Yao, Q.Z. Sheng, A.H. Ngu, J. Yu, A. Segev, Unified collaborative and content-based web service recommendation, IEEE Trans. Serv. Comput. 8 (3) (2015) 453–466.
[7] G. Tian, J. Wang, K. He, C. Sun, Y. Tian, Integrating implicit feedbacks for time-aware web service recommendations, Inf. Syst. Front (2015) 1–15.
[8] O. Khalid, M.U.S. Khan, S.U. Khan, A.Y. Zomaya, Omnisuggest: a ubiquitous cloud-based context-aware recommendation system for mobile social networks, IEEE Trans. Serv. Comput. 7 (3) (2014) 401–414.
[9] L. Chen, J. Wu, H. Jian, H. Deng, Z. Wu, Instant recommendation for web services composition, IEEE Trans. Serv. Comput. 7 (4) (2014) 586–598.
[10] Qi Zhang, Lu Cheng, Raouf Boutaba, Cloud computing: state-of-the-art and research challenges, J. Internet Serv. Appl. 1.1 (2010) 7–18.
[11] S. Taneja, K. Rathi, A trust evaluation model to recommend a service provider to a customer in cloud environment, Int. J. Comput. Appl. 121 (2) (2015).
[12] Bhaskar Prasad Rimal, Eunmi Choi, Ian Lumb, A taxonomy and survey of cloud computing systems, INC, IMS and IDC, 2009, pp. 44–51.
[13] R.B. Rodrigues, C.M. da Silva, W.O. Ferreira, G. MM, A Cloud-based recommendation system, International Conference on Www/Internet, 2013, pp. 384–386.

[14] M. Zhang, R. Ranjan, S. Nepal, M. Menzel, A. Haller, A declarative recommender system for cloud infrastructure services selection, International Conference on Grid Economics and Business Models, Springer Berlin Heidelberg, 2015, November, pp. 102–113.
[15] R. Pereira, H. Lopes, K. Breitman, V. Mundim, W. Peixoto, Cloud based real-time collaborative filtering for item–item recommendations, Comput. Ind. 65 (2) (2014) 279–290.
[16] G. Cheng, L. Liu, H. Lei, Z. Lin, M. Li, T. Yu, QoS-prediction cloud service recommendation by collaborative filtering in cloud manufacturing platform, Adv. Inf. Sci. Serv. Sci. 5 (3) (2013) 295.
[17] X. Zheng, L. Da Xu, S. Chai, QoS recommendation in cloud services, IEEE Access 5 (2017) 5171–5177.
[18] H. Ma, Z.G. Hu, User preferences-aware recommendation for trustworthy cloud services based on fuzzy clustering, J. Cent. S. Univ. 22 (2015) 3495–3505.
[19] Q. Yu, CloudRec: a framework for personalized service Recommendation in the Cloud, Knowl. Inf. Syst. 43 (2) (2015) 417–443.
[20] T.Q. Jiang, W. Lu, Improved slope one algorithm based on time weight, Applied Mechanics and Materials, vol. 347, Trans Tech Publications, 2013, pp. 2365–2368.
[21] Y.M. Afify, I.F. Moawad, N.L. Badr, M.F. Tolba, A personalized recommender system for SaaS services, Concurrency Comput. Pract. Ex. (2016), http://dx.doi.org/10.1002/cpe.3877.
[22] M.B. Amin, S. Hussain, M. Han, B.H. Kang, Y.Y. Ik, S. Jun, S. Lee, Profiling-based energy-aware recommendation system for cloud platforms, Computer Science and its Applications, Springer Berlin Heidelberg, 2015, pp. 851–859.
[23] J. Wheal, Y. Yang, CSRecommender: a cloud service searching and recommendation system, J. Comput. Commun. 3 (06) (2015) 65.
[24] S. Yan, C. Chen, G. Zhao, B.S. Lee, Cloud service recommendation and selection for enterprises, 2012 8th International Conference on Network and Service Management (Cnsm) and 2012 Workshop on Systems Virtualiztion Management (Svm), IEEE, 2012, October, pp. 430–434.
[25] S. Meng, Z. Zhou, T. Huang, D. Li, S. Wang, F. Fei, ... W. Dou, A temporal-aware hybrid collaborative recommendation method for cloud service, Web Services (ICWS), 2016 IEEE International Conference on, IEEE, 2016, June, pp. 252–259.
[26] R. Wille, Restructuring lattice theory: an approach based on hierarchies of concepts, International Conference on Formal Concept Analysis, Springer Berlin Heidelberg, 2009, May, pp. 314–339.
[27] C.A. Kumar, P.K. Singh, Knowledge representation using formal concept analysis: a study on concept generation, Glob. Trend. Intell. Comput. Res. Develop. (2014) 306–336.
[28] G. Fenza, S. Senatore, Friendly web services selection exploiting fuzzy formal concept analysis, Soft Comput. 14 (8) (2010) 811–819.
[29] A. Formica, Semantic web search based on rough sets and fuzzy formal concept analysis, Knowl. Base Syst. 26 (2012) 40–47.
[30] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[31] Y. Shi, Ranking and Context-awareness in Recommender Systems, TU Delft, Delft University of Technology, 2013.
[32] C.J. Willmott, K. Matsuura, Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance, Clim. Res. 30 (1) (2005) 79–82.
[33] Z. Zheng, H. Ma, M.R. Lyu, I. King, Wsrec: a collaborative filtering based web service recommender system, Web Services, 2009. ICWS 2009. IEEE International Conference on, IEEE, 2009, July, pp. 437–444.
[34] Z. Zheng, H. Ma, M.R. Lyu, I. King, Qos-aware web service recommendation by collaborative filtering, IEEE Trans. Serv. Comput. 4 (2) (2011) 140–152.
[35] Y. Hu, Q. Peng, X. Hu, R. Yang, Web service recommendation based on time series forecasting and collaborative filtering, Web Services (ICWS), 2015 IEEE International Conference on, IEEE, 2015, June, pp. 233–240.
[36] F. Aznoli, N.J. Navimipour, Cloud services recommendation: reviewing the recent advances and suggesting the future research directions, J. Netw. Comput. Appl. 77 (2017) 73–86.
[37] M. Unuvar, S. Tosi, Y.N. Doganata, M.G. Steinder, A.N. Tantawi, Selecting optimum cloud availability zones by learning user satisfaction levels, IEEE Trans. Serv. Comput. 8 (2) (2015) 199–211.
[38] Z. Ou, H. Zhuang, J.K. Nurminen, A. Ylä-Jääski, P. Hui, Exploiting hardware heterogeneity within the same instance type of Amazon EC2, HotCloud, 2012.
[39] H. Ma, Z. Hu, Cloud service recommendation based on trust measurement using ternary interval numbers, Smart Computing (SMARTCOMP), 2014 International Conference on, IEEE, 2014, November, pp. 21–24.
[40] S. Soltani, P. Martin, K. Elgazzar, QuARAM recommender: case-based reasoning for IaaS service selection, Cloud and Autonomic Computing (ICCAC), 2014 International Conference on, IEEE, 2014, September, pp. 220–226.
[41] S. Taneja, K. Rathi, A trust evaluation model to recommend a service provider to a customer in cloud environment, Int. J. Comput. Appl. 121 (2) (2015).
[42] M. Zhang, R. Ranjan, S. Nepal, M. Menzel, A. Haller, A declarative recommender system for cloud infrastructure services selection, International Conference on Grid Economics and Business Models, Springer Berlin Heidelberg, 2015, November, pp. 102–113.
[43] K. Jayapriya, N.A.B. Mary, R.S. Rajesh, Cloud service recommendation based on a correlated QoS ranking prediction, J. Netw. Syst. Manag. (2015) 1–28.
[44] C.A. Kumar, S.C. Mouliswaran, P. Amriteya, S.R. Arun, Fuzzy formal concept analysis approach for information retrieval, Proceedings of the Fifth International Conference on Fuzzy and Neuro Computing (FANCCO-2015), Springer International Publishing, 2015, pp. 255–271.
[45] C. De Maio, G. Fenza, V. Loia, S. Senatore, Hierarchical web resources retrieval by exploiting fuzzy formal concept analysis, Inf. Process. Manag. 48 (3) (2012) 399–418.
[46] A. Formica, Semantic web search based on rough sets and fuzzy formal concept analysis, Knowl. Base Syst. 26 (2012) 40–47.
[47] R.C. Chen, C.T. Bau, C.J. Yeh, Merging domain ontologies based on the WordNet system and fuzzy formal concept analysis techniques, Appl. Soft Comput. 11 (2) (2011) 1908–1923.
[48] C.M. Subramanian, A.K. Cherukuri, C. Chelliah, Modeling fuzzy role based access control using fuzzy formal concept analysis, International Symposium on Security in Computing and Communication, Springer International Publishing, 2015, August, pp. 176–185.
[49] J. Poelmans, S.O. Kuznetsov, D.I. Ignatov, G. Dedene, Formal concept analysis in knowledge processing: a survey on models and techniques, Expert Syst. Appl. 40 (16) (2013) 6601–6623.
[50] W. Yiqin, Q. Zhenji, L. Cailin, Virtual machine scheduling in cloud computing based on formal concept analysis, Comput. Digital Eng. 42 (12) (2014).
[51] Koitz, I. T., & Glinz, M. A Fuzzy Galois Lattices Approach to Requirements Elicitation for Cloud Services.
[52] J. Xu, W. Gong, Y. Wang, A cloud service discovery approach based on FCA, 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, vol. 3, IEEE, 2012, pp. 1357–1361.
[53] H. Ferchichi, J. Akaichi, Using mapreduce for efficient parallel processing of continuous K nearest neighbors in road networks, J. Software Syst. Dev. (2016), http://dx.doi.org/10.5171/2016.356668.
[54] H. Ma, Z. Hu, Recommend trustworthy services using interval numbers of four parameters via cloud model for potential users, Front. Comput. Sci. 9 (6) (2015) 887–903.
[55] D. Ardagna, G. Casale, M. Ciavotta, J.F. Pérez, W. Wang, Quality-of-service in cloud computing: modeling techniques and their applications, J. Internet Serv. Appl. 5 (1) (2014) 11.
[56] Movileanu, B., Sacarea, C., & Sotropa, D. F. Fact-a tool for temporal formal concept analysis. In 14th International Conference on Formal Concept Analysis (p. 95).
[57] Carmen De Maio, Giuseppe Fenza, Vincenzo Loia, Mimmo Parente, Time aware knowledge extraction for microblog summarization on twitter, Inf. Fusion 28 (2016) 60–74.