Cluster Computing (2024) 27:2845–2874
https://doi.org/10.1007/s10586-023-04119-2

Improved clustering-based hybrid recommendation system to offer personalized cloud services

Hajer Nabli[1] · Raoudha Ben Djemaa[1] · Ikram Amous Ben Amor[2]

Received: 4 June 2023 / Revised: 25 July 2023 / Accepted: 27 July 2023 / Published online: 23 August 2023

Abstract
The ever-increasing number of cloud services has led to the service's identification problem. It has become difficult to provide users with cloud services that meet their functional and non-functional requirements, especially as many cloud services offer the same or similar functionality but with different execution constraints (cloud characteristics, QoS, price, and so on). Service recommendation systems can solve the service's identification problem by helping users to retrieve the right cloud services according to their desired needs. However, the majority of service recommendation systems rely on user feedback to locate the user's neighbors, predict missing ratings, and rank the recommended services. As a result, users' rating histories might cause three major problems: cold start, data sparsity, and malicious attack. In order to deal with these issues, we propose in this paper a hybrid recommendation approach, called "HRPCS", that provides a list of personalized cloud services to the active user. This approach is based on user and service clustering. In this approach, cloud services are recommended based on the user's needs (functional and non-functional) and QoS preferences. Then, the services are ranked according to their prices and credibility. Further, the proposed approach returns a list of diversified cloud services. The experimental results confirmed our expectations and proved the effectiveness of our approach.

Keywords Hybrid recommendation · Cloud services · Personalized cloud services · Clustering · QoS preferences · Diversity

1 Introduction

Cloud Computing is a widely adopted paradigm that offers various services on-demand [1]. Consequently, cloud providers seek to deliver powerful and reliable cloud services to their users on a pay-as-you-go basis. At the same time, users (individuals or businesses) are using cloud services to meet their needs. However, with the growing number of available cloud services, many services offer the same or similar functionality, which makes it challenging to provide users with services that meet their requirements exactly. Therefore, finding the relevant cloud services that exactly meet users' functional and non-functional needs remains at the heart of a lot of research. In the literature, researchers used sophisticated methods and algorithms to help users evaluate, select, and rank the best services from a large pool of available alternatives. These methods have been divided into logic-based [2, 3], ontology-based [4–7], multi-criteria decision-making (MCDM)-based [8–12], and optimization-based [13, 14] cloud service selection approaches. These studies, however, mainly depend on Quality of Service (QoS) measures to objectively assess and rank cloud services based on their performance [15] and do not take into account user-specific preferences, historical usage patterns, or interactions among users.

A more efficient solution is to use recommendation systems. In fact, recommendation systems are designed to provide personalized recommendations based on user-specific data and interactions. This personalization ensures that the recommended cloud services are tailored to each user's unique needs, leading to higher user satisfaction and a more efficient selection process. In general, recommendation systems adopt three main techniques: (i) Content-based approaches [16] recommend services that are similar

✉ Hajer Nabli
nabli.hajer@yahoo.fr

[1] Higher Institute of Computer Science and Communication Technologies of Hammam Sousse, University of Sousse, Sousse, Tunisia

[2] National School of Electronics and Telecommunications of Sfax, University of Sfax, Sfax, Tunisia

to the cloud services previously used by the active user, (ii) Collaborative filtering-based approaches [17] identify the active user's neighbors and predict the missing values of the user-service matrix based on similar users' previous behavior, and (iii) Hybrid approaches [18] that are a combination of content and collaborative methods. The collaborative filtering-based technique is the most popular and commonly used technique for recommender systems [19, 20]. Most collaborative filtering-based approaches, on the other hand, rely on users' rating (or feedback) to find the user's neighbors, predict missing ratings, and rank the recommended services. However, users' rating histories allow for the detection of three key issues: cold start, data sparsity, and malicious attack. The cold start problem arises when the system is unable to establish any relations between users and services for which it lacks sufficient data. In other terms, the cold start problem occurs when a new user has not rated a sufficient number of services (user's cold start problem), or when a new service has not been rated by a significant number of users (service's cold start problem) [21]. In this instance, the recommendation algorithm will be unable to provide accurate recommendations since it will be unable to locate an appropriate neighborhood for the user or the service in order to properly predict the missing data. The data sparsity problem occurs when a small subset of available services are rated. Hence, the rating matrix used for recommendation is rather sparse. As a result of this, identifying similar users or services becomes difficult. Consequently, the similarity between two users or services cannot be calculated and eventually, the accuracy of prediction becomes extremely low [21]. The malicious attack problem arises when users create malicious profiles that contain biased and deceptive feedback in order to sway the active user's decision and influence the ranking of recommendations in their favor [22]. As a result, user satisfaction declines and recommendation accuracy falls to low levels.

To cope with the above-mentioned problems, we present, in this paper, "HRPCS", a hybrid recommendation of personalized cloud services approach. The novelty of our research lies in the development of an enhanced clustering-based hybrid recommendation approach that effectively addresses the challenge of providing personalized cloud services to users. By combining clustering techniques with recommendation algorithms, our proposal leverages the advantages of both approaches to deliver accurate and personalized recommendations. Indeed, our proposed approach is based on user and service clustering. A key factor in accurate clustering is the proper determination of similar users and services. As a result, we presented a novel similarity measure for users based on their QoS preferences, location, and usage traces, as well as a similarity measure for services based on their location and usage traces. These equations are also valid for new users and new services where we can maintain the users' QoS preferences and location and the services' location, and therefore solve the user and service cold start problems. In addition, we adopted the clustering method to reduce the computational complexity and address the data sparsity issue. Furthermore, integrating the elimination of users' feedback into recommender systems offers a promising approach to tackling the pressing issue of malicious attacks. By removing users' feedback, particularly from unverified or suspicious sources, the system can significantly reduce the impact of certain types of attacks. Notably, shilling attacks, profile injection attacks, and data poisoning attacks, which heavily rely on manipulating feedback to promote certain items and influence recommendations, can be mitigated. The absence of maliciously crafted feedback prevents the infiltration of misleading data into the training process, thus fostering a more secure and trustworthy recommendation environment. Indeed, the absence of user-generated feedback helps to neutralize the influence of fake positive ratings and manipulated preferences, preventing attackers from distorting the system's recommendations. Moreover, by minimizing its reliance on user data, the recommender system can reduce its vulnerability to privacy breaches and user-targeted exploits. Additionally, we focus on ranking the services based on their prices and credibility. The credibility of each service is calculated based on the expected QoS parameter values (declared by the service provider) and observable QoS parameter values (specific measurements related to the performance of a service that can be easily observed, monitored, and quantified). Finally, our approach is founded on the concept of diversity. This diversity allows for less redundancy in the list of recommendations while also taking into account the diverse interests of users.

The motivation behind this study stems from the increasing demand for personalized cloud services, where users expect recommendations that align with their specific requirements. By offering improved recommendations, the proposed approach not only enhances user satisfaction but also contributes to the optimization of resource allocation and utilization in cloud environments. This paper aims to make a significant impact on the field by presenting an innovative solution that addresses the challenges of personalization in cloud service recommendations. Indeed, we are interested in contributing the following in this paper in order to design an approach that effectively recommends personalized cloud services:

- A clustering-based hybrid recommendation approach is presented for delivering personalized cloud services that match the user's needs (functional and non-functional) and QoS preferences.

- To improve user satisfaction, personalized services are ranked using a scoring formula that incorporates two criteria: the pricing and the credibility of the services.
- Our approach is built on diversity and generates a diverse list of cloud services, offering a list of suggestions without redundancy.
- Troubleshooting: Cold Start, Data sparsity, and Malicious attack.

The remainder of this paper is constructed as follows. In Sect. 2, we present the related work on cloud service recommendation. Section 3 presents the proposed Hybrid Recommendation of Personalized Cloud Services approach. A discussion of the results are presented in Sect. 4. Finally, Sect. 5 concludes our paper and gives an outlook on possible future research directions.

2 Related works

In this section, we present and discuss some related work on cloud service recommendation.

2.1 Cloud service recommendation approaches

Soltani et al. [23] presented a platform for IaaS service selection named QuARAM Service Recommender. This platform adopts case-based reasoning to choose the cloud infrastructure service that best matches the requirements of the user's application. The recommendation process begins when the user submits their cloud application specification to the platform. From this specification, a set of information will be extracted, namely, application requirements, user preferences, and application deployment entities. Then, the system searches for similar cases using three knowledge bases (Application Case Base, Adaptation Case Base, and Vendor Knowledge Base) and returns a list of recommended deployment configurations to the user. However, there are drawbacks in this solution that can be summarized in five points: (1) Among the three types of IaaS, SaaS, and PaaS services, this system supports only the IaaS services, (2) It may not be suitable for non-expert users as the service model must be built using the TOSCA specification, (3) The QoS parameters are not specified in this work, (4) This system does not use the clustering model to handle the large search space problem, and (5) This work does not solve the diversity problem.

Afify et al. [24] proposed, SaaS Recommender (SaaSRec), a personalized recommendation system for SaaS services that is based on reputation. In SaaSRec, user feedback is used to calculate service reputation values. In addition, users are grouped into communities using the Hierarchical Agglomerative Clustering algorithm (HAC) which is associated with a similarity measure based on the location, interests, and feedback of the users. Indeed, the recommendation process goes through several phases to find the optimal set of recommended services. In the first phase, SaaSRec recovers the SaaS services that meet the functional needs of the user. In the second phase, the system filters the recovered services according to the requirements of QoS and service characteristics specified by the user. Then, the collaborative filtering approach uses user communities to retrieve local reputation values from the recommended list of services. The content-based filtering approach considers the metadata of the services to calculate the similarity between the recommended list of services and the user's profile (location, interests, preferences, and previous selections of services). Finally, similar services with profile, reputation values, and prices are combined to rank the list of recommended services. However, this measure has shortcomings that can be summarized in three points: (1) The recommended services are limited to SaaS services, (2) The cloud characteristics covered are limited to common characteristics (such as license type, payment system, formal agreement), where the specific characteristics of SaaS services are ignored, and (3) The problems of new service's cold start, data sparsity and diversity are not solved in this work.

Balaji and Rajkumar [25] proposed a cloud service recommendation system based on hybrid collaborative filtering. In this work, the K-means clustering method of users is associated with the similarity calculation (using Cosine similarity) before evaluating the prediction of ratings (feedback) and finally, producing the recommendation. However, this system presents some drawbacks that can be summarized in the four following points: (1) Only the user feedback are used as service recommendation criteria (lack of cloud characteristics and QoS parameters), (2) The authors assume that all the services have the same functionality, (3) The services recommended for the user are not ranked, and (4) The system does not provide solutions for a new user's cold start, malicious attack and diversity.

Ding et al. [26] presented a method based on collaborative filtering for predicting historical data (eg. QoS values) and recommending cloud services with better user satisfaction. First, similar users are identified using an enhanced similarity measure (eKRCC) which is based on Kendall Rank Correlation Coefficient (KRCC) and Jaccard's coefficient. Next, the authors proposed a satisfaction function to determine user expectations regarding the quality of cloud services. Finally, the top-k similar users with their satisfaction values are used to make predictions of missing QoS values and subsequently select the appropriate cloud services. However, this approach has some drawbacks that we summarize in the following four points:

(1) There is a lack of information about the cloud characteristics, (2) This work does not address the problem of large search space, (3) The services recommended to the active user are not ranked, and (4) The cold start, data sparsity, malicious attack, and diversity problems are not solved in this work.

Ma et al. [27] proposed a cloud service recommendation system sensitive to QoS variations. The service recommendation process begins when an active user submits their request, which includes their functional and non-functional requirements, to the system. First, the candidate services, which meet the user's functional requirements, are returned. Next, a method of identifying user neighbors is presented to support QoS prediction through collaborative filtering. In this work, QoS parameters are monitored, collected periodically, and stored in four QoS models (central tendency, variation range, frequency of variation, and period). The Mahalanobis distance is used to measure the similarity of the QoS patterns. Finally, the proposed method is formulated as a multi-criteria decision-making problem, and an improved TOPSIS method is exploited to solve it, taking into account both the variation of the QoS and the preferences of the active user during different periods. Nevertheless, this solution has some shortcomings that can be summarized as follows: (1) This work does not cover other important criteria, such as cloud characteristics, (2) The large amount of data collected is not properly addressed in this work, (3) Services are ranked according to the users' QoS while other criteria, such as service price and provider credibility, had better be considered, and finally, (4) The data sparsity, malicious attack, and diversity problems are not solved in this work.

Mezni et al. [28] proposed a recommendation system based on collaborative filtering for cloud services using Fuzzy Formal Concept Analysis (CR-FFCA) to generate reliable recommendations using fuzzy lattices. This method transformed the cloud service repository into a set of small clusters, in which the relationships between high-quality services and users with the highest ratings (feedback) are organized using formal concepts. The fuzzy lattices representation helps to exclude unnecessary data related to the cloud. This involves excluding poorly rated cloud services from the candidate service set and eliminating less similar users who share a few common services with the active user. However, this method has some drawbacks that we summarize in the five following points: (1) Cloud characteristics are not considered in the recommendation process, (2) The computation of similar users depends only on the number of services in common while ignoring other important criteria, such as the users' location and the QoS preferences, which can improve the cluster's quality. In other words, this solution is unable to identify users similar to a new user who has no usage history (due to lack of neighbors), therefore, it will be unable to offer recommendations to this user, (3) In this work, the cold start problem for new services is solved by eliminating them from the recommendation process although they may be relevant services for the active user, (4) The missing data are considered to be useless and therefore are excluded, and (5) Feedback from malicious users are not processed.

Wang et al. [29] proposed a cloud service recommendation approach based on collaborative filtering by exploring user history. This approach first calculates user similarity using an improved cosine similarity method, which is adjusted based on the popularity of the cloud service. Then, several similar users are selected as neighbors of the active user. Finally, this system predicts the possibility that the active user invokes new (non-invoked) services based on neighboring users. However, this system has some deficiencies that can be summarized in the following four points: (1) The proposed solution neglects several other criteria that can improve the accuracy of the service recommendation, such as the cloud characteristics and the QoS, (2) This work has not solved the problem of large search space, (3) The proposed method is not efficient for new users besides, newly released cloud services that do not have usage records are unlikely to be recommended, and (4) The issues of data sparsity, malicious attack and diversity are not addressed in this work.

Djiroun et al. [30] proposed an approach to recommend cloud services using clustering methods to give better visibility to users and services. The proposed solution is based on two aspects: the analysis of the content and description of the services, and the analysis of user behavior, that is to say, the interactions and previous actions with the services. In this context, two types of recommendation approaches are combined. A content-based recommendation and a collaborative-based recommendation, which aims to predict the interests of the active user on services not previously consulted or not used, based on the analysis of similar users' traces (i.e. their feedback and behavior on the services). Finally, a pruning process is applied to the recommended services resulting from the two recommendation approaches to eliminate the services that are not suitable for the active user. However, this approach has some drawbacks which can be summarized in the following three points: (1) Cloud services are QoS sensitive due to the dynamic cloud environment while the proposed solution neglects the impact of the QoS on the service recommendation, (2) Other important criteria are ignored in the computation of the clusters (users and services), such as their location, their usage history (for users and services) and the users' QoS preferences, which can improve the quality of the clusters, and (3) The problems of cold start, malicious attack and diversity are not addressed in this work.

Zheng et al. [31] proposed an approach based on collaborative filtering to recommend cloud services. This system has three steps and works as follows. In the first step, the Spearman coefficient is adopted to calculate the similarity between the active user and the other users. In the second step, the k-nearest neighbors' technique is used to find a set of neighbors that are similar to the active user. Finally, a user-based collaborative filtering approach is adopted to predict active user's QoS feedback for unrated services and determine a set of recommended services for the active user. However, this technique has some limitations that we summarize in the following five points: (1) Cloud characteristics are not taken into account in this solution, (2) The recommendation is based on a single QoS criterion, (3) This system does not provide a solution to reduce the search space, such as clustering, (4) The problems of cold start, data sparsity, malicious attack and diversity are not solved, and (5) The authors assume that all the available services have the same functionality.

Nagarajan et al. [32] presented a broker-based context-aware recommendation system with QoS factors for IaaS type of cloud services. The proposed broker extracts the service details based on their contextual data. PMF model and matrix factorization methods are improved to increase the performance of broker in QoS prediction. Nevertheless, this solution has some shortcomings that can be summarized as follows: (1) The recommended services are limited to IaaS services, (2) This work does not cover other important criteria, such as cloud characteristics, neighbors of the user and the service, usage history. (3) This work does not provide a solution to reduce the search space, such as clustering, (4) Services are ranked according to the highest QoS values while other criteria, such as service price and provider credibility, should be taken into account as well, and (5) The new user's cold start, data sparsity, malicious attack, and diversity issues are not solved in this work.

Ngaffo et al. [33] proposed a data sparsity service recommendation approach that aims to predict relevant cloud services for end-users. First, the QoS prediction of the current time is performed using a factorization matrix technique. Thereafter, the QoS prediction of the future time interval is performed using a time series forecasting method based on an AutoRegressive Integrated Moving Average (ARIMA) model. However, this system has some drawbacks that we summarize in the following points; (1) The cloud characteristics, user's and service's neighbors, user's QoS preferences, and usage history are not defined, (2) The proposed method cannot provide a solution to reduce the search space, such as clustering, (3) The services recommended to the active user are not ranked, and (4) The cold start, malicious attack, and diversity issues are not addressed in this work.

2.2 Discussion

Nowadays, cloud services are characterized by their increasing volume, variety, and heterogeneity, which makes finding relevant cloud services tailored to user needs a real challenge. In this regard, cloud users must have a system to find and recommend services that best meet their preferences, among a large number of existing choices. There are several works in the literature that deals with this subject. Their objectives are to filter the services for each active user to meet their needs (functional and non-functional) and their QoS preferences. The comparison between the different proposed approaches is established according to the following challenges (See Table 1).

- Cold Start: indicates whether the approach addresses the cold start problem for a new user and a new service.
- Data Sparsity: indicates whether the approach was able to reduce the data sparsity problem.
- Malicious Attack: indicates whether the approach is immune to untrusted users.
- Diversity: indicates whether the approach offers diversified recommendations to users.

Table 1 presents the comparative analysis of recommendation approaches according to the different challenges mentioned above. We note that only some approaches [23, 27, 28] managed to solve the cold start problem. This problem affects both new users and new services that are introduced into the system. Besides, recommendation systems suffer from the data sparsity problem when the number of services evaluated by users is too low compared to the total number present in the system. This problem affects the ability of the system to recommend all available services and the accuracy of the recommendations generated. Only [25, 28, 30, 33] who solved the problem of data sparsity either by reducing the search space using the clustering method [25, 28, 30] or by applying a flexible matrix factorization technique [33]. In addition, recommendation systems are vulnerable to malicious attack. These attacks are caused by malicious users who want to manipulate and redirect the recommendation to their needs by providing misleading feedback, high or low. Approaches based on collaborative filtering are vulnerable to attacks because they allow their users to participate in the recommendation calculation of their neighbors, without their explicit permission. Only the solution of [24] solved the problem of malicious attack by introducing a user score incremented by 0.1 for each objective feedback. On the other hand, none of the approaches proposed in the literature offers a diversified recommendation although this criterion is highly requested and appreciated by users since it makes it possible to solve the problem of over-

Table 1 Challenges of recommendation systems

Approach	Cold start		Data sparsity	Malicious attack	Diversity
	User	Service			
[23]	✓	✓	✓	N/A	✗
[24]	✓	✗	✗	✓	✗
[25]	✗	✓	✓	✗	✗
[26]	✗	✗	✗	✗	✗
[27]	✓	✓	✗	✗	✗
[28]	✓	✓	✓	✗	✗
[29]	✗	✗	✗	✗	✗
[30]	✗	✗	✓	✗	✗
[31]	✗	✗	✗	✗	✗
[32]	✗	✓	✗	✗	✗
[33]	✗	✗	✓	✗	✗

specialization and also improve the quality of the user history with the recommendation system.

To highlight the added value of our approach compared to its predecessors, we indicate in Table 2 the different properties that a recommendation system must verify. The comparison is made according to the following properties:

- Service Type: This property describes the service types covered by the proposed approach (IaaS, PaaS, SaaS, XaaS).
- Recommendation Technique: This property defines the technique used to recommend cloud services. The filtering techniques generally used for calculating the recommendation are either content-based filtering, collaborative filtering, or hybrid filtering.
- Recommendation Criteria: This property describes the set of criteria covered by the proposed approach. These criteria are fundamental in recommending the best cloud services.
- Clustering: This property indicates whether the proposed solution relies on clustering methods to reduce search space for users and services.
- Ranking: This property indicates whether the proposed solution returns a ranked list of services by specifying the method and the criteria used.

By studying the literature, we notice that most of the approaches offer recommendations for the three types of cloud services, namely IaaS, PaaS and SaaS [25–31, 33]. In addition, we note that QoS plays an important role to perform a personalized recommendation [24, 26–28, 31–33]. These approaches, on the other hand, merely recommend cloud services with the best feedback or values for a specific QoS parameter, without taking into account the user's QoS preferences, which is required to satisfy the user and provide a personalized recommendation.

Besides, to reduce the search space, narrow the data sparsity problem and accelerate the process of finding services, it is necessary to have powerful tools allowing the processing of data collections that are available in increasing quantity. Unsupervised Data Mining methods such as clustering methods are a response to this need. Indeed, clustering aims to partition large volumes of data into a set of data groups (clusters) with regard to their similarities. We note that the work [30] used clustering algorithms to group not only users into clusters, but also services in order to improve the recommendation process. While other approaches are based only on users clustering [24, 25, 28]. However, the clustering criteria used in these approaches which define similarity between users or services are imperfect. Thus, important criteria are ignored, such as location, usage history for both users (services previously used by users) and services (users who have used the service) and user's QoS preferences, which can improve the quality of clusters.

In addition, ranking is one of the fundamental problems in the field of research on cloud services. The problem is to sort the services retrieved by relevance. As a result, most of the studied approaches [23, 24, 27–32] recommended ranked services according to several criteria. In general, the vast majority of users tend to look for cloud services that guarantee both the best price and the best service credibility. Only the work of [24] which takes into account the price to classify the services recovered. On the other hand, in order to improve user satisfaction, it is essential to take into account the periodic variation of QoS. However, it is important to note that none of the existing approaches have addressed the calculation of the service's credibility according to the variation of the QoS.

Table 2 Comparison of cloud service recommendation approaches

Approach	Service type	Recommendation technique	Recommendation criteria	Clustering		Ranking	
				Users	Services	Method	Criteria
[23]	IaaS	Case-based reasoning	Configuration parameters	✗	✗	Score	–
[24]	SaaS	Hybrid filtering	QoS + Cloud characteristics + Reputation + User profile	HAC (Location + interests + Feedbacks)	✗	Fuzzy ranking	User profile + Reputation + Price
[25]	XaaS	Collaborative filtering	Feedbacks	K-means (Feedbacks)	✗	✗	✗
[26]	XaaS	Collaborative filtering	QoS	✗	✗	✗	✗
[27]	XaaS	Collaborative filtering + TOPSIS	QoS	✗	✗	TOPSIS + Mahalanobis distance	QoS
[28]	XaaS	Collaborative filtering	QoS + Feedbacks	Fuzzy lattice (Number of common services)	✗	Service score	Feedbacks
[29]	XaaS	Collaborative filtering	Usage history	✗	✗	–	Predicted possibility
[30]	XaaS	Hybrid filtering	User interests (Feedbacks + Behavior)	K-Medoids (User profile)	K-Medoids (Cloud characteristics)	–	User interests
[31]	XaaS	Collaborative filtering	QoS + Feedbacks	✗	✗	Prediction function	Feedbacks
[32]	IaaS	Matrix factorization	Service context + QoS	✗	✗	Highest QoS values	QoS
[33]	XaaS	Flexible matrix factorization + ARIMA model	QoS	✗	✗	✗	✗

3 Proposed approach

The purpose of our personalized recommendation approach is to improve user satisfaction by recommending ranked and diversified cloud services that match their preferences and are likely to be of interest to them. To achieve this goal, we propose the architecture described in Fig. 1. In addition to the user interface, the proposed architecture is made up of three modules, namely, "Clustering Module", "Hybrid Recommendation Module", and "Ranking and Diversification Module". It also includes three databases, namely "Historical Data", "Cloud Service Registry", and "Monitoring Data".

Initially, the user initiates his request by specifying a set of data via a graphical interface, namely his functional requirements (service type and category), his non-functional requirements (cloud characteristics and payment mode), his QoS preferences as well as his priorities for the ranking criteria ①. In fact, two interfaces are used to recommend personalized cloud services. We provide the user with a first interface that lists the various cloud service types along with the categories that they fall under. The user merely needs to choose the service's type and category. In the instance shown in Fig. 2, the user is looking for an IaaS cloud service that falls under the compute category. The user then confirms his decision. Following the user's entry of his functional requirements, a second interface prompts him to indicate his non-functional requirements (such as cloud characteristics and payment mode), QoS preferences, and the ranking criteria's order of importance. Figure 3 illustrates an example of parameters specified according to the user's choice.

The "Hybrid Recommendation Module" starts as soon as it receives the request from the user ②. In fact, This module is made up of three phases. First, the "Recovery of used services" phase makes it possible to query the "Clustering Module" in order to return the first list of recommendations (2^a). This list is used for the activation of

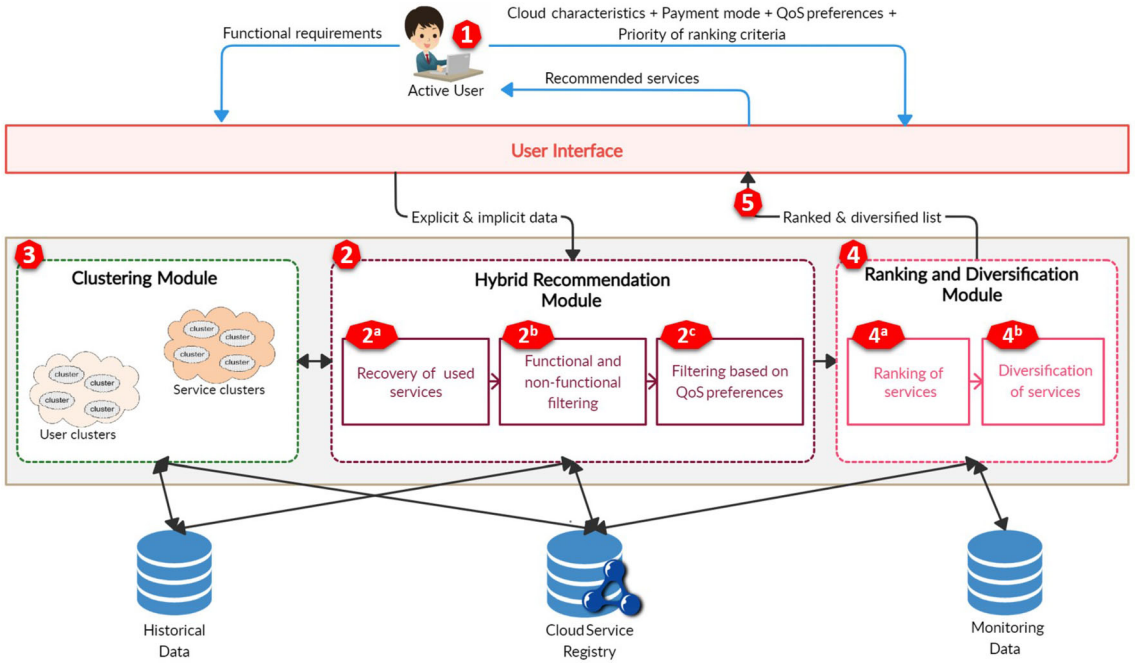


Fig. 1 Architecture of "HRPCS" approach

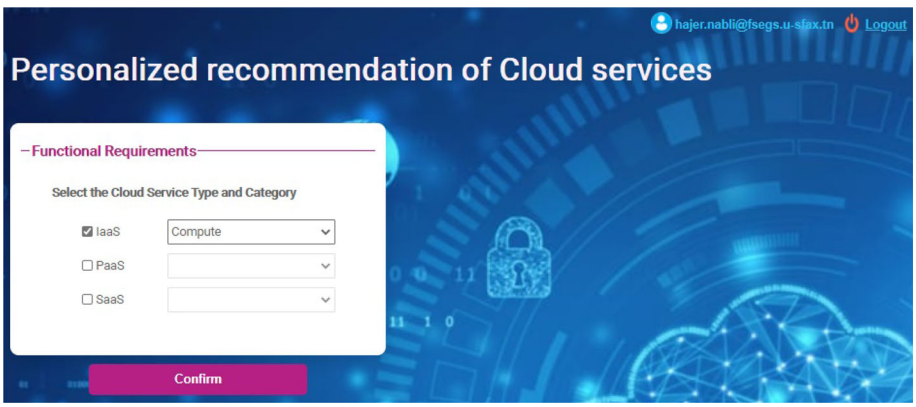


Fig. 2 Selection of the service's type and category

the "Functional and non-functional filtering" phase in order to refine the first list and only return the cloud services that meet the user's functional and non-functional needs (2^b). Finally, the "Filtering based on QoS preferences" phase is used to recommend to the active user a list of services compatible with his QoS preferences (2^c). The "Clustering Module" ③ is based on the application of a hierarchical clustering technique in order to group users and services into clusters according to their proposed similarity measures (we come back to these similarity measures in more detail in Sect. 3.1). The clusters are calculated offline to be used by the "Hybrid Recommendation Module" in order to deduce the cluster to which a user or a service belongs as needed. Finally, the journey ends by querying the "Ranking and Diversification Module" which takes as input the results from the "Hybrid Recommendation Module" ④. Two phases are proposed in this module, namely "Ranking of services" (4^a) and "Diversification of services" (4^b) in order to output a final list of cloud services which are ranked and diversified ⑤. In the following, we detail the different modules presented in the proposed architecture.

As a motivating example, think about a situation where a user wishes to hunt for cloud services utilizing our "HRPCS" approach. The user is prompted to enter his non-functional needs after entering his functional requirements (type and category). These non-functional requirements include cloud characteristics, payment mode, QoS preferences, and the priorities of the ranking criteria. The user

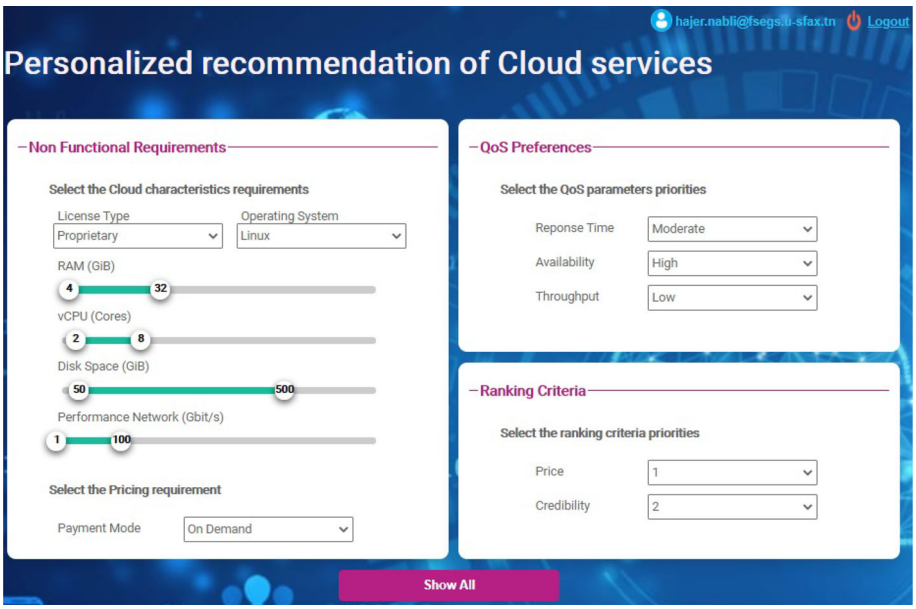


Fig. 3 Selection of non-functional requirements, QoS preferences, and ranking priorities

requirements that are both functional and non-functional are shown in Table 3.

3.1 Clustering module

This work proposes a hybrid recommendation approach that joins the user clustering technology and the service clustering technology. The "Clustering Module" consists of calculating the similarities between the objects (users or services), then the objects are grouped into clusters so that the objects of the same cluster are more similar to each other than to those of other clusters. This module is used in order to deal with the problem of sensitivity to missing data (data sparsity) and the problem of scaling up by reducing the neighbor search space and consequently improving the calculation time of recommendations as well as their accuracy. As illustrated in Fig. 4, the "Clustering Module" is composed of two phases, namely "Calculation of similarity measures" and "Generation of clusters". In fact, the phase of "Calculation of similarity measures" consists of two steps. The "Similarity between users" is the initial step that establishes a new similarity measure for users based on QoS preferences, location, and user usage traces (services previously used). As well as the second step "Similarity between services", which aids in providing a similarity measure for services based on location and service usage traces (users who have used the service). In addition, we will need the "Historical Data" database and the "Cloud Service Registry" to calculate these similarities correctly. The Agglomerative Hierarchical Clustering (AHC) approach is then utilized to build user and service clusters in the "Generation of clusters" phase.

The similarity measures proposed for identifying similar users and similar services, as well as the clustering algorithm, are described in detail in the following sections.

Table 3 User's functional and non-functional requirements

Functional		Non-functional			
Type	Category	Cloud Characteristics	Payment Mode	QoS Preferences	Ranking Criteria
IaaS	Compute	LT: Proprietary OS: Linux RAM: 4GB to 32GB vCPU: 2 cores to 8 cores DS: 50GB to 500GB PN: 1Gbit/s to 100Gbit/s	OD	RT: Moderate AV: High THP: Low	PR: 1 CRD: 2

LT license type, *OS* operating system, *DS* disque space, *PN* performance network, *OD* on demand, *RT* response time, *AV* availability, *THP* throughput, *PR* price, *CRD* credibility

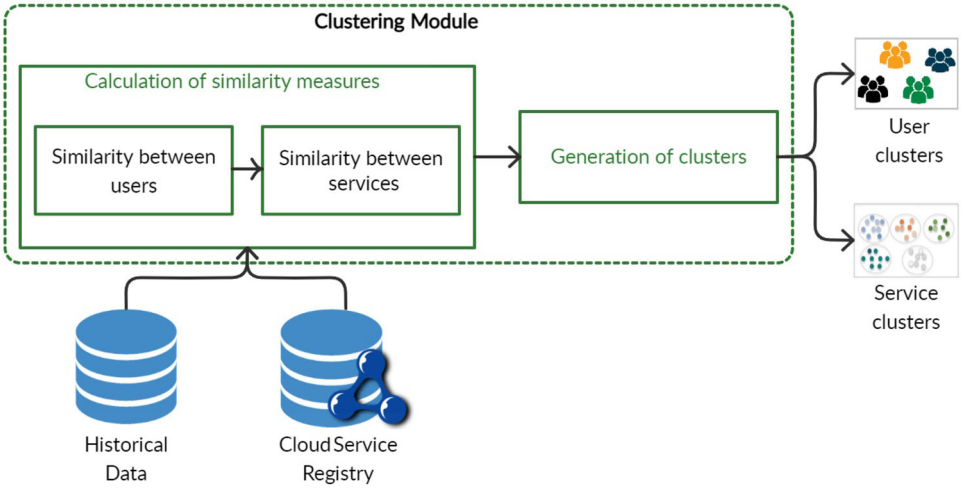


Fig. 4 Phases of "Clustering Module"

3.1.1 Calculation of similarity measures

The correct assessment of similar users and similar services is a significant aspect in accurate clustering. As a result, we propose new user and service similarity measures.

3.1.1.1 Similarity between users To calculate the similarity between two users, we define a similarity measure that is a linear combination between three similarity aspects, namely, QoS preference similarity, location similarity, and trace similarity. As for QoS preferences, they are entered by the user using a graphical interface. For the user location, it is derived from their IP address. As for the usage traces, they represent all the services previously used by the active user and which are retrieved from the "Historical Data" database. This module uses SQL queries to search for usage traces in the "Historical Data" database. In the case of a new user (without usage traces), only the QoS preferences and the location are used for users' similarity calculation. In this way, we can avoid the problem of new user's cold start. Therefore, the more the two users tend to request the same QoS preferences, be in the same place, and use the same services, the more similar they will be. The proposed similarity between users u and v is calculated as the sum of these three weighted similarities, as shown in Eq. 1:

$$\begin{aligned} Sim_{users}(u,v) = {} & w_{Upref} * Sim_{Upref}(u,v) \\ & + w_{Uloc} * Sim_{Uloc}(u,v) \\ & + w_{Utrc} * Sim_{Utrc}(u,v) \end{aligned} \tag{1}$$

where Sim_{Upref}, Sim_{Uloc} and Sim_{Utrc} respectively represent similarities in QoS preferences, location and usage traces, and w_{Upref}, w_{Uloc} and w_{Utrc} represent their weights, respectively. The weights are used to adjust the relevance of the three aspects of similarities and they were assigned based on their recommendation accuracy (the weight values are proven in Sect. 4). We present in what follows the three aspects of similarity used in the measurement of similarity between users.

QoS preference similarity. After accessing a graphical interface, the user chooses their QoS preferences by assigning a rating to each QoS parameter (exp, High, Moderate, or Low). The QoS parameters reflect the cloud service properties, including response time, availability, reliability, security, throughput, and so on. The received ratings are then normalized and transformed into real values between 0 and 1, whose sum is always equal to 1 (exp, High = 0.6, Moderate = 0.3, and Low = 0.1). More formally, QoS preferences are defined as a weight vector $P_u = \{w_{u,1}, w_{u,2}, \ldots, w_{u,m}\}$, where each weight $w_{u,j} \geq 0$ denotes the importance of the QoS parameter of index j with respect to user u and $\sum_{j=1}^{m} w_{u,j} = 1$. Therefore, it suffices to compare the weight vectors P_u and P_v to determine the similarity of QoS preferences between two users u and v. To do this, we use the Cosine similarity, as shown in Eq. 2:

$$Sim_{Upref}(u,v) = \frac{\sum_{j=1}^{m} w_{u,j} * w_{v,j}}{\sqrt{\sum_{j=1}^{m} w_{u,j}^2}\sqrt{\sum_{j=1}^{m} w_{v,j}^2}} \tag{2}$$

where $w_{u,j}$ and $w_{v,j}$ respectively represent the weights assigned to the QoS parameter of index j by user u and user v, and m is the number of QoS parameters.

Location similarity. Intuitively, two users are geographically similar if they are in the same geographic location or nearby. In fact, the objective of our choice to integrate location similarity is that the physical locations of users have a significant effect on the variance of the QoS [34–36]. In fact, location similarity is determined by first determining the physical distance between users. In fact, to translate his IP address into a geographic location (latitude

and longitude), we made advantage of the free GeoLite database provided by the MaxMind API.[1] Then, the Haversine formula (Eq. 3) is used to calculate the distance between two locations l_1 and l_2 on the Earth surface specified in latitude and longitude.

$$Dist(l_1, l_2) = 2 * R * \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta lat}{2}\right) + \cos(lat_1) * \cos(lat_2) * \sin^2\left(\frac{\Delta lng}{2}\right)}\right) \tag{3}$$

Knowing that, $\Delta lat = lat_2 - lat_1$ $\Delta lng = lng_2 - lng_1$ where R is the radius of the Earth ($R = 6371km$), lat_1 and lat_2 are the latitudes of l_1 and l_2 in radians and lng_1 and lng_2 are the longitudes of l_1 and l_2 in radians.

Since the similarity is inversely proportional to the distance, in other words, a short distance indicates a high similarity while a long-distance means a low similarity. Then, the location similarity between users u and v is calculated using Eq. 4:

$$Sim_{Uloc}(u, v) = 1 - \frac{Dist(u, v)}{\arg\max_{i \in E(u^*)} Dist(u, i)} \tag{4}$$

where $E(u^*)$ is the set of users who have used services and who are stored in the "Historical Data" database, $Dist(u, v)$ is the Haversine distance which is used to calculate the geographic distance between two users u and v whose locations are specified in latitude and longitude (Eq. 3).

Trace similarity. Trace similarity represents the association between two users regarding their use of the same services. In other words, two users are considered similar if they have previously invoked a number of common cloud services. Our choice to integrate trace similarity ensures that the set of similar users is more homogeneous because they share the same interests and concerns [37]. The similarity of traces is calculated using the following Jaccard formula:

$$Sim_{Utrc}(u, v) = \frac{|S_u \cap S_v|}{|S_u| + |S_v| - |S_u \cap S_v|} \tag{5}$$

where S_u and S_v respectively represent the set of services used by user u and the set of services used by user v, $S_u \cap S_v$ is the subset of services that user u and user v have commonly used. The set of services used by a user is retrieved from the "Historical Data" database.

3.1.1.2 Similarity between services The similarity between two services is calculated using a similarity measure that linearly combines two aspects of similarity, namely, location similarity and trace similarity. Regarding the location of services, it is obtained either from the "Cloud Service Registry" in the event of a new service, or from the "Historical Data" database in the case of a service already in use. For the usage traces, they represent all the users who used the service and who are retrieved from the "Historical Data" database. In the case of a new service (no information about the users who used this service is available), only the location is used for the service similarity calculation. In this way, we will be able to solve the problem of new service's cold start. In fact, the more the two services tend to be in the same location and to be used by the same users, the more similar they will be. As illustrated in Eq. 6, the proposed similarity between the services x and y is calculated as the sum of these two weighted similarities:

$$Sim_{services}(x, y) = w_{Sloc} * Sim_{Sloc}(x, y) + w_{Strc} * Sim_{Strc}(x, y) \tag{6}$$

where Sim_{Sloc} and Sim_{Strc} represent the similarities in location and usage traces, respectively, and w_{Sloc} and w_{Strc} represent their weights, respectively. The weights are used to adjust the relevance of the two aspects of similarity and have been assigned based on their recommendation precision (more details in Sect. 4). In the following, we present the two aspects of similarity used in the measure of similarity between services.

Location similarity. To identify which services are geographically similar, we must first find their locations. Indeed, the services having the same location can have similar performances [35, 36]. There are then two types of services, those called new services (they are stored in the "Cloud Service Registry" and have never been used) and those called used (they have traces of use in the "Historical Data" database). In the case of a new service, its physical address is retrieved from the "Cloud Service Registry". Then, this address will be converted to a geographic location (latitude and longitude) using Google's Geocoding API.[2] Table 4 shows examples of address conversion. For the used services, the location in latitude and longitude is obtained from the "Historical Data" database. Therefore, the location similarity between services x and y is calculated using Eq. 7:

$$Sim_{Sloc}(x, y) = 1 - \frac{Dist(x, y)}{\arg\max_{i \in E(s^*)} Dist(x, i)} \tag{7}$$

where $E(s^*)$ is the set of services in the "Cloud Service Registry", $Dist(x, y)$ is the Haversine distance (Eq. 3) which is used to calculate the geographic distance between

[1] https://www.maxmind.com.

[2] https://developers.google.com/maps/documentation/geocoding/overview

Table 4 Address conversion

Services	Physical address	Geographical location
Amazon EC2	US West (Los Angeles)	lat:34.0522342, lng:−118.2436849
Oracle Cloud Platform	Canada Southeast (Toronto)	lat:43.653226, lng:−79.3831843
Overleaf Service	UK	lat:55.378051, lng:−3.435973

two services x and y whose locations are specified in latitude and longitude.

Trace similarity. Two services are considered similar if they have previously been used by a number of common cloud users. Therefore, to calculate the trace similarity between two services, it is necessary to find the set of users who used each service, then to associate them using Jaccard's formula. Indeed, we consider that the services having users in common can have similar performances [37]. The trace similarity between services x and y is calculated using Eq. 8:

$$Sim_{Strc}(x,y) = \frac{|U_x \cap U_y|}{|U_x| + |U_y| - |U_x \cap U_y|} \tag{8}$$

where U_x and U_y respectively represent the set of users who used the service x and the set of users who used the service y, $U_x \cap U_y$ is the subset users who have used both x and y services. This set of users is retrieved from the "Historical Data" database.

In the next section, we explain the clustering algorithm used to group users and services into clusters, respectively, using the proposed user similarity measure (Eq. 1) and service similarity measure (Eq. 6).

3.1.2 Generation of clusters

Regarding user clustering and service clustering, we propose to use the Agglomerative Hierarchical Clustering (AHC) algorithm thanks to its hierarchical structure, flexibility in cluster shape, and interpretability. AHC does not require a predefined number of clusters, and its dendrogram visualization allows users to explore relationships at different levels of granularity. It is suitable for distance-based clustering on various types of data, making it applicable to numerical, categorical, and mixed datasets. Generally, an agglomerative approach begins by assigning each item (user or service) to a separate cluster or singleton. Then, it calculates the inter-cluster similarities (distance inter-clusters) for each pair of clusters, thus building a similarity matrix. The main step in the agglomerative algorithm lies in the choice of the method that calculates the inter-cluster similarities. We think that by including the similarity measures we provided for users and services in the Average-Linkage method, we are able to achieve this aim. The similarity between two clusters in the Average-Linkage grouping process is the average of the similarities between all pairs of elements (users or services), such as the first element is in a cluster and the other element is in another cluster. The two clusters with the highest average similarity are merged to form a new cluster. The Average-Linkage method defines the similarity between two clusters C_k and C_l by Eq. 9:

$$Sim_{AL}(C_k, C_l) = \frac{1}{|C_k||C_l|} \sum_{e_i \in C_k} \sum_{e_j \in C_l} Sim_*(e_i, e_j) \tag{9}$$

where $Sim_*(e_i, e_j)$ is replaced by $Sim_{users}(u, v)$ (Eq. 1) in case of user clustering and by $Sim_{services}(x, y)$ (Eq. 6) in the case of service clustering.

At the end of this module, users and services are grouped into clusters according to their proposed similarity measures. The clusters are computed offline for use by the "Hybrid Recommendation Module" to provide personalized service recommendations for the active user.

3.2 Hybrid recommendation module

In this section, we propose a new hybrid recommendation approach, named "HRPCS". Our work is globally different from the approaches studied in the state-of-the-art mainly on two aspects. First, the user feedback are often used in recommendation systems, especially those using collaborative filtering, to rank services or to find the closest neighbors of the active user and predict missing ratings. However, this kind of approach is normally fragile to malicious attack [22]. These attacks are generated by users who introduce malicious profiles consisting of biased feedback to affect the recommendations ranking and manipulate the active user decision. In this regard, we will eliminate the utilization of user feedback and instead concentrate on predicting unavailable QoS values. This consideration takes into account the possibility that services with the same location and accessed by the same users could showcase analogous performance [35–37]. By adopting this strategy, we aim to minimize the potential for malicious attacks. Second, most of the approaches studied in the state-of-the-art recommend services with the best values of a certain QoS parameter without exploiting the QoS preferences of the active user. Indeed, a user's QoS preferences are certainly important for a personalized recommendation of cloud services.

The "Hybrid Recommendation Module" is divided into three phases, as shown in Fig. 5, which are "Recovery of used services", "Functional and non-functional filtering", and "Filtering based on QoS preferences". This module is triggered by a user's request in which he specifies his functional and non-functional requirements as well as his QoS preferences via a graphical interface. The user and service clusters, the "Historical Data" database, and the "Cloud Service Registry" are all required to complete the process of this module. The "Recovery of used services" phase is the first that is executed. It will cover services utilized by users who are similar to the active user. Based on the user's request, his explicit, implicit, and historical data are obtained in order to create the user's profile. After that, we computed the similarities between the user's profile and the previously created user clusters to decide which cluster he belongs to. The services utilized by similar users are then extracted. This is the first list of cloud services in our hybrid recommendation process. The "Functional and non-functional filtering" phase is used to reduce the number of services received from the first list by considering the user's functional needs, which are represented by the cloud service type and category. The services are then filtered according to the user's non-functional requirements, such as cloud characteristics and payment mode. Finally, this phase delivers a second list of services that satisfy the active user's functional and non-functional needs. At last, during the "Filtering based on QoS preferences" phase, a list of services suitable with the user's QoS preferences is recommended.

The different phases of the "Hybrid Recommendation Module" are detailed in the following sections.

3.2.1 Recovery of used services

This phase allows the recovery of services used by the user's neighbors. It is based on the Demographic filtering technique that is an alternative to overcome the problem of new user's cold start. This phase is done in three steps: Data retrieval, Similarity computation, and Services extraction, as shown in Fig. 6.

3.2.1.1 Data retrieval This step consists of collecting the relevant data to build the user's profile. The data entered directly by the user via a graphical interface (called explicit) partly contributes to the construction of the user profile. Implicit data (implicitly retrieved during user authentication such as location) and historical data (usage traces or services previously consumed by the user) are also retrieved to complete the user's profile. Then, the information in the user's profile, and more precisely the user's QoS preferences, location, and usage traces (services previously used), is used to calculate the similarities to the user clusters (already built) using the Average-Linkage method (Eq. 9) to determine the cluster to which the active user belongs.

Explicit data. The explicit data represents the information directly provided by the user. These data involve demographic data if it is a new user, the type and category of the sought cloud service, cloud characteristics, payment mode, QoS preferences, and the priority of ranking criteria (service price and credibility).

Implicit data. The implicit information is obtained automatically from the environment in which the user is located. In our work, we are interested in knowing the user's location during the recommendation request. This is

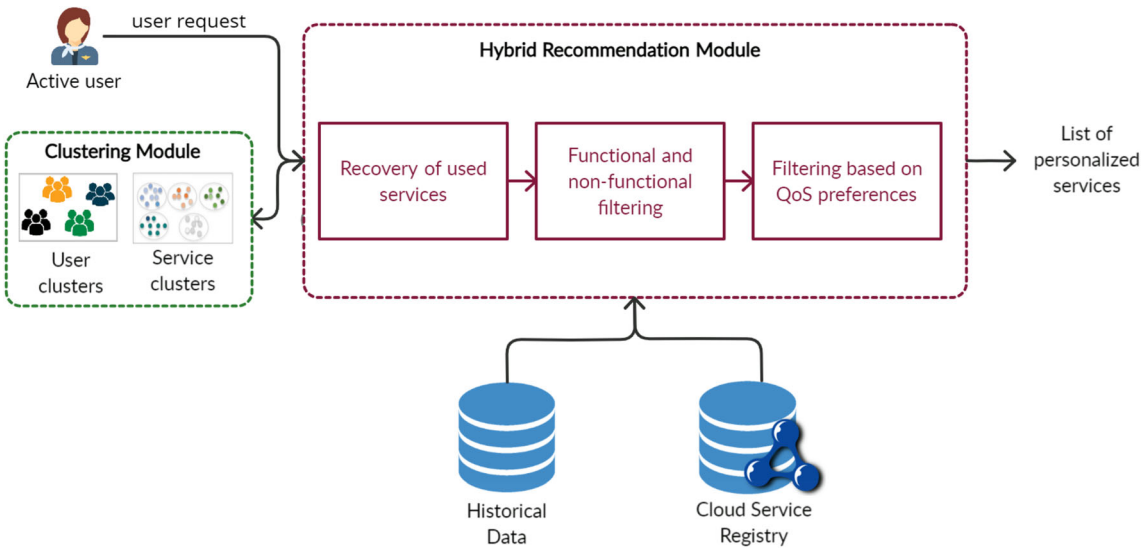


Fig. 5 Phases of "Hybrid Recommendation Module"

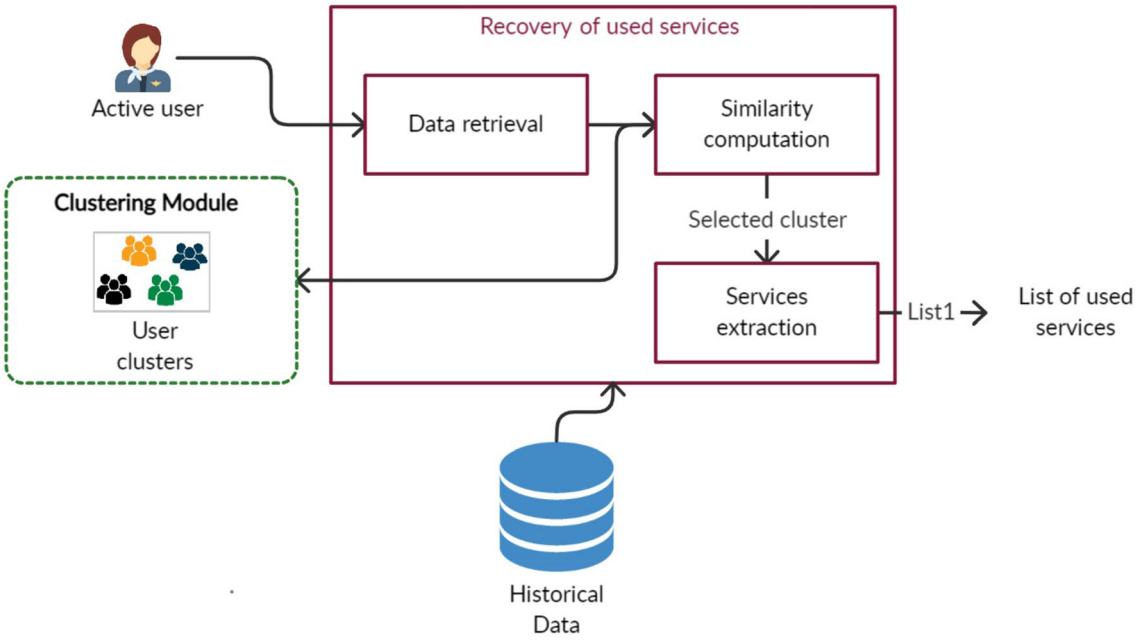


Fig. 6 Steps of the "Recovery of used services" phase

done by capturing the user's IP address and convert it into a geographic location (latitude and longitude).

Historical data. Historical information is obtained using a historical data database. Indeed, to allow the storage of usage traces, we have developed and maintained an SQL database. This database contains users' demographic data, users and services locations, and the users' QoS preferences requested previously. The "Historical Data" database is queried to insert (if a new user) or recover (if a user has usage traces) data that can be used by our approach. Such data can be represented by the list of services used previously or the list of users who have used a given service. The class diagram, represented by Fig. 7, contains the list of classes that constitute the database of "Historical Data".

- **User**: This class represents a user. In particular, it contains the user's personal information, such as name, address, e-mail, and others.
- **Service**: This class contains the identifier of the cloud service used by a user.
- **Consumption**: This class represents an n-ary association between a user and a service. It stores the QoS preferences required by a user for a cloud service and the user's location when using the service. We can have several instances of the *Consumption* association-class linking the same user to the same service.
- **Location**: This class describes information relating to the geographic location of users and services (latitude and longitude).

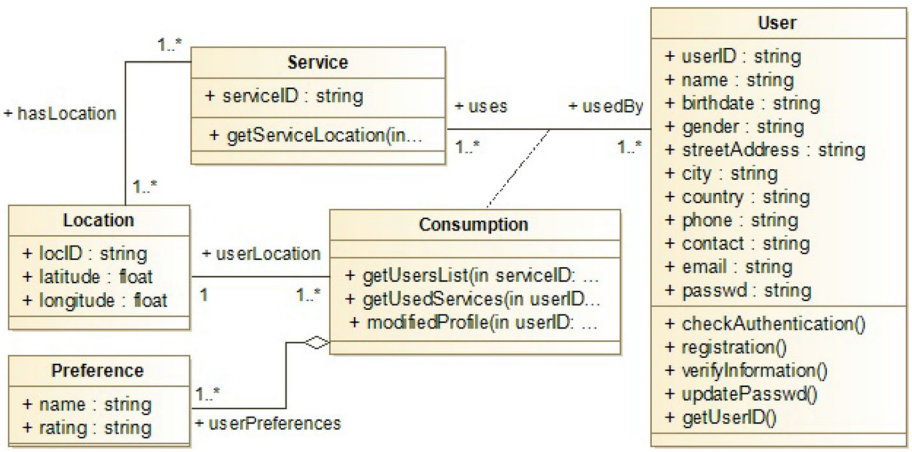


Fig. 7 Schema of the "Historical Data" database

- **Preference**: This class represents the QoS preferences claimed by users. Its properties are the QoS parameter name and the rating assigned to this parameter.

Going back to the earlier example, the user's profile is created using the logged-in user's explicit, implicit, and historical data (See Fig. 8).

3.2.1.2 Similarity computation Once the data constituting the user's profile is identified, certain data is analyzed (QoS preferences, location, and usage traces) to determine the cluster to which the active user belongs. The objective is to identify users who have the same data as the active user. There are two types of users: those called new users and those considered as existing users (they have usage traces in the "Historical Data" database).

In the case of a new user without history, his QoS preferences and his location are only used for the similarity calculation (Eq. 1). This solves the new users' cold start problem. Thus, his similarities with user clusters built previously (by querying the "Clustering Module") are

```
<?xml version="1.0" encoding="UTF-8"?>
<user_profile>
        <personal_details>
                <userID>u192314536</userID>
                <name>Hajer Nabli</name>
                ⋮
                <location>
                        <ipAddress>197.14.158.184</ipAddress>
                        <latitude>36.79800033569336</latitude>
                        <longitude>10.171699523925781</longitude>
                </location>
        </personal_details>
                ⋮
        <requirements>
            <functional>
                    <type>IaaS</type>
                    <category>Compute</category>
            </functional>
            <non_functional>
                    <cloud_characteristics>
                            <license_type>Proprietary</license_type>
                            <os>Linux</os>
                            <ram>
                                    <min>4</min>
                                    <max>32</max>
                            </ram>
                            <vcpu>
                                    <min>2</min>
                                    <max>8</max>
                            </vcpu>
                            <disque_space>
                                    <min>50</min>
                                    <max>500</max>
                            </disque_space>
                            <performance_network>
                                    <min>1</min>
                                    <max>100</max>
                            </performance_network>
                    </cloud_characteristics>
                    <payment_mode>On Demand</payment_mode>
            </non_functional>
            <qos_preferences>
                    <reponse_time>Moderate</reponse_time>
                    <availability>High</availability>
                    <throughput>Low</throughput>
            </qos_preferences>
            <ranking_criteria>
                <price>1</price>
                    <credibility>2</credibility>
            </ranking_criteria>
        </requirements>
        <historical_data>
                <service>serv1a319828-18ab</service>
                <service>serv4f5880d6-0425</service>
                <service>serv9cc5edf6-b131</service>
        </historical_data>
</user_profile>
```

Fig. 8 An XML file describing the user's profile

calculated based on the Average-Linkage method (Eq. 9). Then, the active user is assigned to the cluster with the highest similarity. Finally, the step of "Services extraction" is triggered immediately after sending the concerned cluster.

For existing users where all data is available, we check if the active user still belongs to the same cluster. This can be explained by changes that the user can make to their QoS preferences and location. So, two cases arise. Either the user has kept the same QoS preferences and the same location and in this case, we directly find the cluster to which he belongs. Either the user has made some changes and in this case, he is considered a new user and we redo the calculation of similarities with the user clusters. Afterward, the user must be assigned to the nearest cluster. As with a new user, the mission of the "Similarity computation" step ends when it sends the cluster to which the active user belongs to the next step "Services extraction".

3.2.1.3 Services extraction It is the last step of the "Recovery of used services" phase. Indeed, once we found the user cluster to which the active user belongs, we can extract the services used by these nearby users. The extracted services represent the first list of cloud services in our hybrid recommendation approach.

Table 5 displays the list of services that have been checked out, using the prior scenario as an example. These offerings are regarded as "Functional and non-functional filtering" phase input data.

Algorithm 1 illustrates the steps for recovering the used services. We note the set of cloud services issued from the "Recovery of used services" phase by $US = \{s_1, s_2, \ldots, s_p\}$. This set is used to activate the second phase of our hybrid recommendation process, namely "Functional and non-functional filtering" which we detail in the next section.

3.2.2 Functional and non-functional filtering

The "Functional and non-functional filtering" phase is triggered when the first list of cloud services (US) is obtained. The second phase applies the knowledge-based filtering technique to obtain a more concise set of services, taking into account the requirements entered by the active user via a graphical interface. Indeed, the user defines a set of functional and non-functional constraints. These constraints are the type and category of cloud service to indicate the functional needs of the user. Moreover, the user defines the cloud characteristics (for example, Operating system, License type, Network performance, and others) and payment mode (for example, On-demand, Reserved, Spot, and others) to indicate their non-functional needs. The "Functional and non-functional filtering" phase is divided into two steps: "Functional filtering" and "Non-functional filtering" (See Fig. 9). Therefore, this phase completes the "Functional filtering" step by returning services that satisfy the user's functional requirements. The services found are then filtered based on the user's non-

Algorithm 1 Recovery of used services phase

Require: $request_u$: The request of the active user u, $C_{users} = \{C_1, C_2, \cdots, C_{N_C}\}$: The user clusters

Ensure: US : The first list of cloud services

1: $US = \emptyset$ // Initialize US to the empty set
2: $u\_profile$ = getUserData($request_u$) // Retrieve user data (explicit, implicit, and historical if possible
3: **if** newUser(u) || modifiedProfile(u,$u\_profile$) **then** // If user u is a new user or he is an existing user who has changed their QoS preferences or location
4: $C_u = \{u\}$ // Create a cluster for user u
5: $C_k = \underset{C_i \in C_{users}}{\arg\max}\ Sim_{AL}(C_u, C_i)$ // Using Equation 9 that does call to Equation 1, we identify the closest cluster C_k to the active user u
6: $C_k = C_k \cup C_u$ // Assign user u to the closest cluster C_k
7: **else**
8: C_k = getUserCluster(u) // Find the cluster to which the user u belongs
9: **end if**
// Retrieve the services that are used by the neighbors of the user u
10: **for** each user $u_i \in C_k$ **do**
11: L = getUsedServices(u_i)
12: $US = US \cup L$
13: **end for**
14: Return US

functional demands in the "Non-functional filtering" step. Finally, the second list of services can be obtained by searching the "Cloud Service Registry" using SPARQL queries. The "Cloud Service Registry" allows the storage of the cloud service description model developed in [38, 39]. As shown in Fig. 9, not only the services from the first phase ("Recovery of used services") are filtered, but also the new services that are stored in the "Cloud Service Registry" and which are never used. This essentially aims at solving the new service's cold start problem that is still in a situation of neglect by the majority of works studied in the literature.

In keeping with our example, Table 6 lists the services that satisfy the functional (IaaS services that offer computing power) and non-functional requirements of the active user (Table 3). Although the *Linode* service hasn't been used, we see that it has been added to the list.

Algorithm 2 describes the principle of the "Functional and non-functional filtering" phase. At this level, we have a second, more precise list of cloud services (CS). This list is used as input for the "Filtering based on QoS preferences" phase.

3.2.3 Filtering based on QoS preferences

The "Filtering based on QoS preferences" phase is activated when the second list of cloud services, *CS*, is obtained. The steps of this phase is illustrated in Fig. 10. This phase aims to recommend services based on the QoS values and not on the user feedback. While user feedback is valuable for improving the accuracy and relevance of recommendations, it can also be exploited by malicious actors to manipulate the system or deceive other users. That is why, we have purposefully chosen to ignore user feedback and instead focus on QoS values. This approach offers several advantages, particularly in mitigating the risk of misleading feedback and potential malicious attacks. By eliminating user feedback, we reduce the possibility of

Algorithm 2 Functional and non-functional filtering phase

Require: US : The list of services issued from the first phase, $u\_profile$: The profile of active user u

Ensure: CS : The second list of cloud services

1: $CS = \emptyset$ // initialize CS to the empty set
2: Connection to the "Cloud Service Registry"
3: $L1 =$ getServices(US,$u\_profile$) // Retrieve, from US, the services that meet functional (type and category) and non-functional (cloud characteristics and payment mode) requirements of the user u
4: $L2 =$ getUnusedServices($u\_profile$) // Retrieve, from "Cloud Service Registry", the services that are never used and which meet the functional and non-functional requirements of the user u
5: $CS = L1 \cup L2$
6: Return CS

Table 5 First list of cloud services

URL	Type	Category
https://www.rackspace.com/	IaaS	Compute
https://www.dropbox.com/	SaaS	Storage
https://www.redhat.com/en/technologies/cloud-computing/openshift	PaaS	Develop, deploy, and manage container-based applications
https://aws.amazon.com/ec2/	IaaS	Compute
https://www.office.com/	SaaS	Office apps
https://azure.microsoft.com/services/virtual-machines/	IaaS	Compute
https://aws.amazon.com/dynamodb	PaaS	DataBase
https://aws.amazon.com/fr/elasticbeanstalk/	PaaS	Deploying and scaling web applications
https://cloud.google.com/sql	PaaS	Database

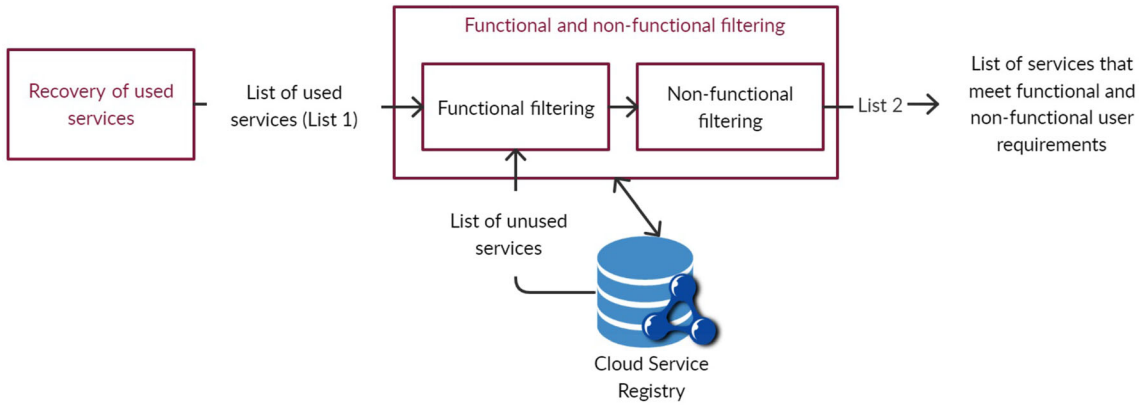


Fig. 9 Steps of the "Functional and non-functional filtering" phase

Table 6 Second list of cloud services

Service	URL	Instances name
cs_1	https://www.rackspace.com/	General1-8
cs_2	https://www.linode.com/	Dedicated 4GB
cs_3	https://azure.microsoft.com/services/virtual-machines/	D2 v3
cs_4	https://aws.amazon.com/ec2/	m5ad.large
cs_5	https://aws.amazon.com/ec2/	r5d.large
cs_6	https://www.linode.com/	Shared 4GB
cs_7	https://www.linode.com/	Dedicated 8GB
cs_8	https://azure.microsoft.com/services/virtual-machines/	D2d v4
cs_9	https://aws.amazon.com/ec2/	m5d.large

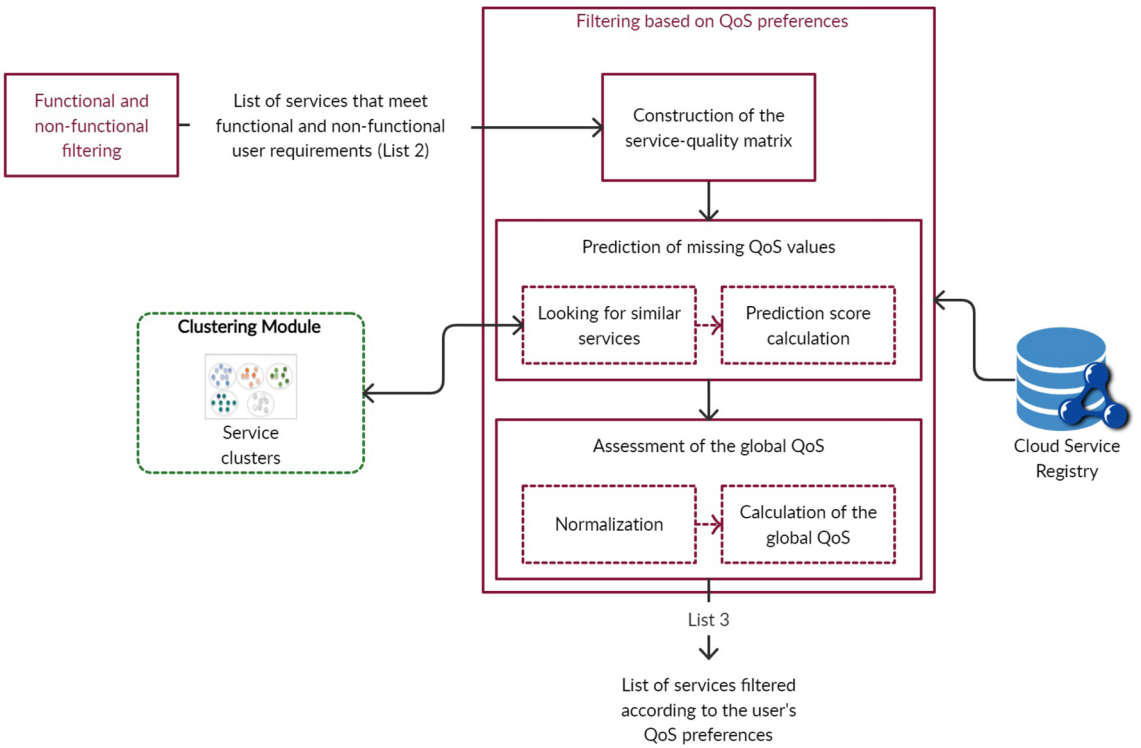


Fig. 10 Steps of the "Filtering based on QoS preferences" phase

intentionally or unintentionally biased or false information that could impact the accuracy of the recommendation system. This proactive measure safeguards the integrity and reliability of the system's performance. Moreover, in this phase, we use collaborative filtering techniques to infer and estimate missing QoS values from the available data, leading to more robust and accurate recommendations. This research direction aligns with the goal of enhancing the trustworthiness and security of recommendation systems, ensuring that users receive reliable and valuable suggestions without compromising their privacy or falling victim to potential attacks.

As a result, instead of preserving a user-service feedback matrix, a new service-quality matrix is formed in the "Construction of the service-quality matrix" step (See Fig. 10).

3.2.3.1 Construction of the service-quality matrix More formally, given, $CS = \{s_1, s_2, \ldots, s_n\}$, the set of n cloud services from the second phase ("Functional and non-functional filtering"). Each service s_i is associated with a line $SQ_i = \{q_{i,1}, q_{i,2}, \ldots, q_{i,m}\}$ in the service-quality matrix, representing its QoS vector where m QoS parameters are used. $q_{i,j}$ represents the jth QoS parameter value. Thus, the service-quality matrix, associating QoS values to the services, is in the following form:

$$Q_{n,m} = \begin{pmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,m} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n,1} & q_{n,2} & \cdots & q_{n,m} \end{pmatrix}$$

In our work, we focus our attention on response time (RT), throughput (THP), and availability (AV) as QoS parameters. In this case, we present below a numerical example of the service-quality matrix that contains the eight cloud services (cs_1 to cs_8) returned by the previous phase (Table 6).

This matrix, on the other hand, may be devoid of many QoS values, resulting in a sparse service-quality matrix. This is why The "Prediction of missing QoS values" is an important step in the service recommendation process. This

$$Q_{9,3} = \begin{array}{c} \\ cs_1 \\ cs_2 \\ cs_3 \\ cs_4 \\ cs_5 \\ cs_6 \\ cs_7 \\ cs_8 \\ cs_9 \end{array} \begin{array}{c} \begin{array}{ccc} RT & THP & AV \end{array} \\ \begin{pmatrix} 0.1 & 0.51 & ? \\ 0.18 & ? & 0.97 \\ 0.24 & ? & 0.99 \\ ? & 0.46 & 0.99 \\ 0.22 & 0.61 & ? \\ ? & 0.45 & 0.95 \\ ? & 0.17 & 0.85 \\ 0.34 & ? & 0.99 \\ ? & 0.62 & 0.98 \end{pmatrix} \end{array}$$

step starts by looking for similar services using the service clusters that have already been calculated. The set of similar services is then utilized to predict the missing QoS value. Finally, we used the "Assessment of the global QoS" step to determine whether or not a cloud service should be recommended. This step consists of calculating the global QoS based on service QoS values and the user's QoS preferences by aggregating these two sets of information. The result of this aggregation provides an overall QoS score for each service. This score represents the global QoS of the service from the perspective of a specific user, considering both the technical performance of the service and the user's QoS preferences. To do that, first, all the QoS parameters' values must be normalized to the same range. Then, the global QoS of the services is calculated. Only the services that are compatible with the user's QoS preferences are kept and considered personalized services.

Consequently, in the cloud service recommendation process, two key difficulties remain to be resolved: the prediction of missing QoS values and the assessment of the global QoS.

3.2.3.2 Prediction of missing QoS values A common hypothesis of existing research is that information from different QoS parameters are assumed to be known and precise. However, in reality, they lack many QoS values, thus forming a sparse service-quality matrix. That is why the prediction of QoS values is a significant step in recommending services. Therefore, if a service has no value for a QoS parameter, we will use similar services to predict the missing value. In this way, we can solve the data sparsity problem.

Looking for similar services. We assume that we are trying to predict the missing QoS value $q_{i,j}$ for the service s_i. So, we start by determining the cluster to which the service s_i belongs using the "Clustering Module". For a good reason, the services that belong to the same location and are used by the same users can have similar performances [35–37]. In addition, our hybrid recommendation process takes into account the independent services that do not belong to any cluster. In this case, the similarities with the service clusters built previously are calculated based on the Average-Linkage method (Eq. 9). Then, the new service is assigned to the cluster with the greatest similarity.

Let C_k be the selected cluster to which the service s_i belongs, and CS is the set of n cloud services, then the list of similar services used for predicting the missing QoS value is the set $SS(s_i) = C_k \cap CS$. However, if $C_k \cap CS = \emptyset$, in this case, we use all the services of the cluster C_k ($SS(s_i) = C_k$).

Prediction score calculation. Now that we have the list of similar services ($SS(s_i)$), we can estimate the missing

QoS value $q_{i,j}$ for the service s_i. The following formula calculates the prediction score:

$$Pred(s_i, q_{i,j}) = \frac{\sum_{s_r \in SS(s_i)} q_{r,j} * Sim_{services}(s_i, s_r)}{\sum_{s_r \in SS(s_i)} Sim_{services}(s_i, s_r)} \quad (10)$$

where $SS(s_i)$ represents the set of similar services, s_r is a service that belongs to the same cluster as s_i, and $q_{r,j}$ represents the jth QoS value that corresponds to the service s_r.

Based on the example of the service-quality matrix, $Q_{9,3}$, presented earlier, we calculate the missing QoS parameter values using the Eq. 10, and the result is shown in the matrix below.

$$Q_{9,3} = \begin{array}{c} \\ cs_1 \\ cs_2 \\ cs_3 \\ cs_4 \\ cs_5 \\ cs_6 \\ cs_7 \\ cs_8 \\ cs_9 \end{array} \begin{array}{c} \begin{array}{ccc} RT & THP & AV \end{array} \\ \begin{pmatrix} 0.1 & 0.51 & 0.9 \\ 0.18 & 0.65 & 0.97 \\ 0.24 & 0.54 & 0.99 \\ 0.2 & 0.46 & 0.99 \\ 0.22 & 0.61 & 0.99 \\ 0.27 & 0.45 & 0.95 \\ 0.3 & 0.17 & 0.85 \\ 0.34 & 0.32 & 0.99 \\ 0.19 & 0.62 & 0.98 \end{pmatrix} \end{array}$$

Another challenge arises that requires a global QoS assessment to take into account the user's QoS preferences.

3.2.3.3 Assessment of the global QoS A user's QoS preferences are certainly important to real service recommendation scenarios, as they can be used to calculate the global QoS of a cloud service in a more precise and personalized way. Thus, we used the global QoS as a basis to judge whether a cloud service can be recommended to a user. From the user point of view, the QoS parameters are generally classified into two categories: Positive parameters and Negative parameters. A QoS parameter is considered positive (respectively, negative) if the higher (respectively, the lower) its value, the keener a user would choose it such as Availability, Throughput (respectively, Response time). After the normalization process, the QoS values will be normalized on the same scale.

Normalization. The normalization step transforms each value of the QoS parameter into a real number between 0 and 1. After the normalization process, a higher value for any QoS parameter means the best quality. Calculating normalization of QoS is given below:

Normalization for Positive QoS parameter

$$q'_{i,j} = \begin{cases} \frac{q_{i,j} - Q_j^{min}}{Q_j^{max} - Q_j^{min}} & \text{if } Q_j^{max} \neq Q_j^{min} \\ 1 & \text{if } Q_j^{max} = Q_j^{min} \, ou \, q_{i,j} = Q_j^{max} \end{cases} \quad (11)$$

Normalization for Negative QoS parameter

$$q'_{i,j} = \begin{cases} \frac{Q_j^{max} - q_{i,j}}{Q_j^{max} - Q_j^{min}} & \text{if } Q_j^{max} \neq Q_j^{min} \\ 1 & \text{if } Q_j^{max} = Q_j^{min} \, ou \, q_{i,j} = Q_j^{min} \end{cases} \quad (12)$$

where the maximum value Q_j^{max} and the minimum value Q_j^{min} of the jth QoS parameter are defined respectively by:

$$\begin{cases} Q_j^{max} = \max(q_{i,j}) \\ Q_j^{min} = \min(q_{i,j}) \end{cases}$$

We note that $Q'_{n,m}$ is the service-quality matrix after the normalization process. It should also be noted that $SQ'_i = \{q'_{i,1}, q'_{i,2}, \ldots, q'_{i,m}\}$ is the QoS vector of the service s_i after the normalization process, in which the QoS parameters values, $q'_{i,j}$, are in the interval [0,1].

$$Q'_{n,m} = \begin{pmatrix} q'_{1,1} & q'_{1,2} & \cdots & q'_{1,m} \\ q'_{2,1} & q'_{2,2} & \cdots & q'_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ q'_{n,1} & q'_{n,2} & \cdots & q'_{n,m} \end{pmatrix}$$

Returning to our example, we present the service-quality matrix after normalization, $Q'_{9,3}$, as follows.

Table 7 Calculation of the global QoS for each cloud service cs_i

Service	Global QoS $GQ(cs_i)$
cs_1	0.59
cs_2	**0.82**
cs_3	**0.80**
cs_4	**0.83**
cs_5	**0.84**
cs_6	0.57
cs_7	0.051
cs_8	0.631
cs_9	**0.84**

$$Q'_{9,3} = \begin{array}{c} \\ cs_1 \\ cs_2 \\ cs_3 \\ cs_4 \\ cs_5 \\ cs_6 \\ cs_7 \\ cs_8 \\ cs_9 \end{array} \begin{array}{c} \begin{array}{ccc} RT & THP & AV \end{array} \\ \begin{pmatrix} 1 & 0.71 & 0.36 \\ 0.67 & 1 & 0.86 \\ 0.42 & 0.77 & 1 \\ 0.58 & 0.6 & 1 \\ 0.5 & 0.92 & 1 \\ 0.29 & 0.58 & 0.71 \\ 0.17 & 0 & 0 \\ 0 & 0.31 & 1 \\ 0.63 & 0.94 & 0.93 \end{pmatrix} \end{array}$$

Calculation of the global QoS. The global QoS of service is calculated with the same method used in [40, 41] in order to retrieve a list of cloud services filtered according to the user's QoS preferences. As explained previously, QoS preferences are entered by the user as ratings (for example, High, Moderate, or Low), which are then normalized and transformed into real values between 0 and 1. Thus, the global QoS $GQ(s_i)$ of the service s_i with m QoS parameters can be calculated by Eq. 13:

$$GQ(s_i) = SQ'_i * P_u^T = \sum_{j=1}^{m} w_{u,j} * q'_{i,j} \quad (13)$$

where $SQ'_i = \{q'_{i,1}, q'_{i,2}, \ldots, q'_{i,m}\}$ is the QoS vector of the service s_i after normalization, $P_u = \{w_{u,1}, w_{u,2}, \ldots, w_{u,m}\}$ represents the user's QoS preferences knowing that $w_{u,j} \geq 0$ denotes the importance of the jth QoS parameter regarding the user u.

After calculating the global QoS for each cloud service $s_i \in CS$, some cloud services will be excluded due to their inconsistency with the user's QoS preferences. In other words, only the services whose global QoS values are greater than a defined threshold,ϵ, are kept. Let $S^* = \{s_i | s_i \in CS \text{ and } GQ(s_i) \geq \epsilon\}$ is the set of candidate services retrieved and sent to the "Ranking and Diversification Module" that we detail in the next section.

Referring back to our example, Table 7 calculates the global QoS of each cloud service $cs_i, i \in [1, 9]$ based on the QoS preferences specified by the active user, which are: availability = high, response time = moderate, and throughput = low (See Fig. 8).

Currently, we select a threshold ϵ of 0.8 (the choice is verified in Sect. 4.4). According to Table 7, the "Filtering based on QoS preferences" phase returns five personalized cloud services that meet the user's needs.

3.3 Ranking and diversification module

Most of the studied approaches focus on recommending a ranked list of cloud services to the user [23, 24, 27–32]. Our objective in this work is not only the recommendation of a list of ranked services but also diversified services. As shown in

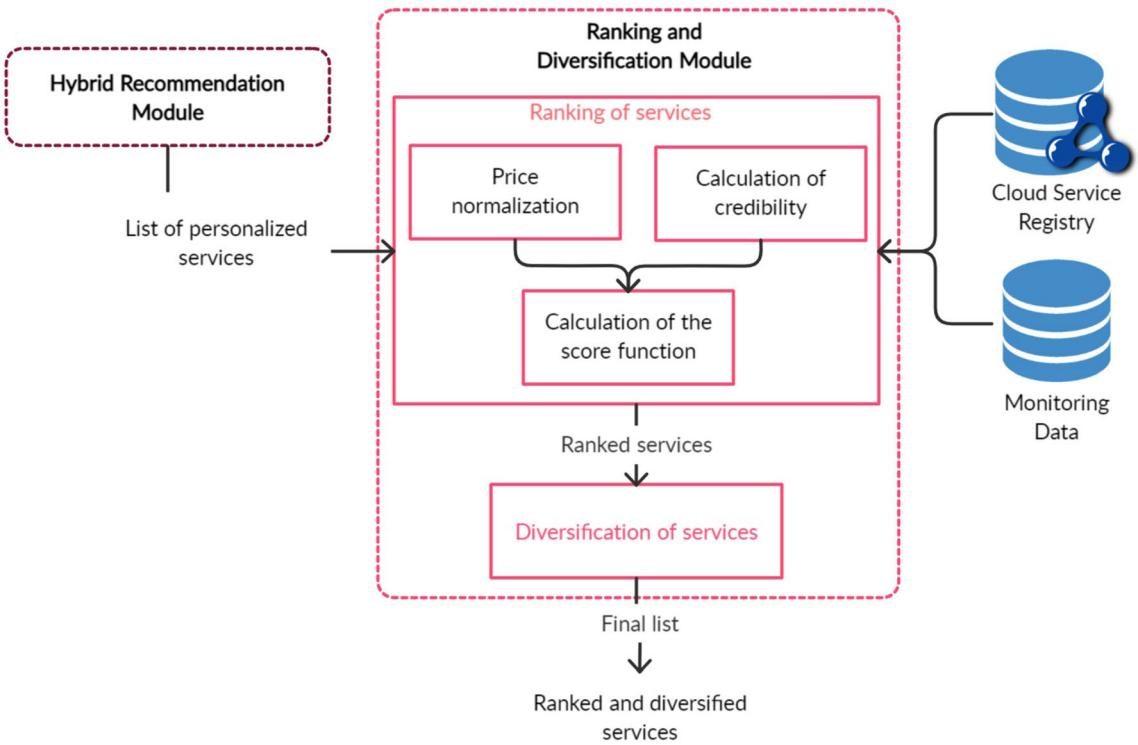


Fig. 11 Phases of the "Ranking and Diversification Module"

Fig. 11, two phases are proposed in this module: "Ranking of services" and "Diversification of services". The first phase aims to return a list of services sorted in descending order using a score function that considers both the service's price and credibility. The second phase returns a list of diversified services. Finally, a final list of ranked and diversified services is recommended to the active user.

3.3.1 Ranking of services

In this phase, we used the Weighted Sum Ranking Algorithm [42]. This algorithm calculates an overall score for each service s_i in the list of candidate services, S^*, by combining two criteria: service *price* and service *credibility*, since they have a significant effect on the users' satisfaction. Each is weighted according to its relative importance. The service with the highest overall score is ranked first, and so on.

3.3.1.1 Price normalization The first criterion is the *Price*, it is the amount that a user must pay to perform the service. We note the price of a service s_i by $price(s_i)$. However, if the price of the service s_i is not between 0 and 1, the expected score may be outside the range of authorized values. Therefore, we must first normalize the price by transforming it into a value varying between 0 and 1. Formally, the normalization of the price of a service s_i is defined by Eq. 14.

$$Prc(s_i) = 1 - \frac{price(s_i)}{\arg\max_{s_l \in S^*} price(s_l)} \tag{14}$$

where S^* is the set of candidate services. The higher the value $Prc(s_i)$, the less the service is cheap. On the other hand, the lower the value $Prc(s_i)$, the more the service is expensive.

3.3.1.2 Calculation of credibility The second criterion is service *Credibility*, which uses QoS parameter monitoring data to assess the service's reliability. Indeed, the difference that may exist between the expected QoS and that observable is an important factor for user satisfaction. In this context, we propose an automatic calculation of the credibility of each service s_i based on the expected values of QoS parameters and the observable values of QoS parameters. Equation 15 illustrates the formula used to calculate the credibility of the service s_i.

$$Crd(s_i) = \frac{\sum_{u_r \in SU(u)} \sum_{j=1}^{m} T_j(u_r, s_i)}{m * |SU(u)|} \tag{15}$$

where $SU(u)$ represents the set of users similar to the active user u, m is the number of QoS parameters, and $T_j(u_r, s_i)$ is the error rate of the jth QoS parameter of the service s_i when it is used by the user u_r. Since the QoS parameters are divided into two categories (positive and negative), so $T_j(u_r, s_i)$ is calculated using the following equation:

$$T_j(u_r, s_i) = \begin{cases} 1 - \left(\dfrac{q_{i,j} - q_{i,j}^{obs}}{q_{i,j}} \right) & \text{if } q_{i,j} \text{ is Positive} \\ 1 - \left(\dfrac{q_{i,j}^{obs} - q_{i,j}}{q_{i,j}^{obs}} \right) & \text{if } q_{i,j} \text{ is Negative} \end{cases} \tag{16}$$

where $q_{i,j}$ is the expected value of the jth QoS parameter of the service s_i ($SQ_i = \{q_{i,1}, q_{i,2}, \ldots, q_{i,m}\}$ is the vector of expected QoS of the service s_i) and $q_{i,j}^{obs}$ is the observable value of the jth QoS parameter of the service s_i ($SQ_i^{obs} = \{q_{i,1}^{obs}, q_{i,2}^{obs}, \ldots, q_{i,m}^{obs}\}$ is the vector of observable QoS of the service s_i), knowing that $q_{i,j}^{obs} = \frac{\sum_{n_q^c} q_{i,j}^c}{n_q^c}$ is the average of the collected values $q_{i,j}^c$ of the jth QoS parameter when using the service s_i by the user u_r and n_q^c is the number of collected values for each QoS parameter. The collected QoS values, $q_{i,j}^c$, are obtained from the "Monitoring Data" database.

After having defined the criteria contributing to the calculation of the score function, we can compute the score of each service. For a user u, the score of the service s_i is defined as follows:

$$Score_u(s_i) = w_p * Prc(s_i) + w_c * Crd(s_i) \tag{17}$$

where $Prc(s_i)$ represents the formula that normalizes the price of the service s_i and $Crd(s_i)$ represents the formula that calculates the credibility of the service s_i. w_p and w_c are the weights used to balance the results of these two formulas. These weights are obtained according to the ranking priority entered by the active user. The candidate services are then ranked in descending order according to their scores. We note the list of ranked services by R_s that is used as input for the second phase of this module.

Continuing with the same example, we initially display a list of cloud services with their scores $Score_u(cs_i)$ listed in descending order. The rankings are based on price (with priority 1 given by the active user) and credibility (with priority 2 given by the active user) of candidate services (See Table 8).

3.3.2 Diversification of services

The objective of existing cloud service recommendation approaches is to recommend the most relevant services to the active user based on their requirements and ranked in order of relevance. However, these approaches assume that all returned services are independent that can lead to many

Table 8 List of ranked cloud services

Service	URL	Instances name	$GQ(cs_i)$	$Score_u(cs_i)$
cs_4	https://aws.amazon.com/ec2/	m5ad.large	0.83	0.48
cs_9	https://aws.amazon.com/ec2/	m5d.large	0.84	0.41
cs_2	https://www.linode.com/	Dedicated 4GB	0.82	0.36
cs_5	https://aws.amazon.com/ec2/	r5d.large	0.84	0.29
cs_3	https://azure.microsoft.com/services/virtual-machines/	D2 v3	0.80	0.21

Table 9 First two services returned to the user

Service	URL	Instances name	$GQ(cs_i)$	$Score_u(cs_i)$
cs_4	https://aws.amazon.com/ec2/	m5ad.large	0.83	0.48
cs_9	https://aws.amazon.com/ec2/	m5d.large	0.84	0.41

Table 10 Diversified cloud services returned to the user

Service	URL	Instances name	$GQ(cs_i)$	$Score_u(cs_i)$
cs_4	https://aws.amazon.com/ec2/	m5ad.large	0.83	0.48
cs_2	https://www.linode.com/	Dedicated 4GB	0.82	0.36

similar cloud services provided by the same vendor appearing in the list of recommendations. Therefore, the user's satisfaction level may decrease due to the redundancy in the recommendations list. For example, if there are a certain number of cloud services provided by the same provider, functionally similar, match the user's requirements, and have a comparatively higher QoS than services from other providers. These services are likely to be at the top of the recommendation list. However, from the user's point of view, the recommended services are redundant and the services of other providers should also be included as much as possible in the list of recommendations. To remove redundancy in the cloud service recommendation list and at the same time improve user satisfaction, diversity should be considered in the recommendation. Thus, diversification in recommendation systems has become one of the main research topics, not only as a way to solve the over-specialization problem but also as an approach to improve the quality of the user experience [43].

Indeed, recommending a set of services that are similar is not as useful for the user, who prefers diversified recommendations [44]. The diversity of the recommendation results makes it possible to best cover the user's interests and allows the discovery of new services. In this context, our proposal takes the form of a service recommendation approach based on diversity. This diversity improves user satisfaction by offering him services in line with his preferences, sufficiently diverse to arouse new interests in him while limiting the excessive redundancy of recommendations.

To better comprehend the diversification phase, let's assume that we will only return to the user the first two services (See Table 9). We see that the cloud services returned are offered by just one provider, *Amazon EC2*. On the other hand, Table 10 shows that another provider (*Linode*) has been added to the list due to the inclusion of diversity.

In order to solve the problem of diversification of results, we use the greedy algorithm [45]. As described in Algorithm 3, the greedy algorithm takes as input the list of ranked services R_s and the diversification threshold κ. On output, the algorithm produces a list of diversified services $D_s \subseteq R_s$, with $|D_s| = \kappa$. Such a list is initialized as an empty set on line 2 and iteratively constructed on lines 3 to 7 of the Algorithm 3. In line 4, $d(s, D_s)$ measures the distance between each service $s \in R_s \backslash D_s$ not yet selected and the services already in D_s, which are selected in the previous iterations of the algorithm. The service with the highest distance, s^*, will be selected, then removed from R_s and added to D_s on lines 5 and 6, respectively. Finally, on line 8, the list of κ diversified services, D_s, produced from the initial list of ranked services, R_s, is returned.

The distance $d(s, D_s)$ between a service $s \in R_s \setminus D_s$ and the list of services D_s is calculated using the distance metric presented in [46].

$$d(s, D_s) = \frac{1 + n_f^{max} - n_{f(s)}}{1 + n_f^{max}} \quad (18)$$

where n_f^{max} represents the maximum number of services offered by the same provider in D_s and $n_{f(s)}$ represents the number of services offered by the service provider s in D_s.

Thus, the "Ranking and Diversification Module" offers a final list of recommendations that is ranked and diversified.

Algorithm 3 Diversification of recommendations using the greedy algorithm

Require: R_s : The list of ranked services, κ : The diversification threshold
Ensure: D_s : The list of diversified recommendations

```
D_s = ∅
while |D_s| < κ do
    s* = arg max_{s∈R_s\D_s} d(s, D_s)      // Select the service with the highest distance
    R_s = R_s \ {s*}                        // Remove the selected service from the set R_s
    D_s = D_s ∪ {s*}                        // Add the selected service to the set D_s
end while
Return D_s
```

4 Evaluation

We describe in this section the results obtained to evaluate our hybrid recommendation approach "HRPCS". This evaluation aims to assess the relevance of the cloud services recommended by our approach and their diversity. To validate our approach, we collected information (type, category, cloud characteristics, price, QoS, and others) from real cloud services from the Internet and published it in the "Cloud Service Registry". The services are used by 339 users whose information, such as the location, has been collected from WS-DREAM,[3] and we have saved them in the "Historical Data" database.

4.1 Evaluation metric

We analyze the experimental data using Precision, Recall, F-measure, and MAE in order to confirm the efficacy of our proposal.

Let's consider the following definitions: *TP* (True Positives) refers to the number of relevant items that are correctly recommended to the user. *FP* (False Positives) refers to the number of irrelevant items that are mistakenly recommended to the user. *FN* (False Negatives) indicates the number of relevant items that are not recommended to the user.

[3] https://wsdream.github.io/

4.1.1 Precision

Precision is a metric that measures the proportion of relevant items (correct recommendations) among all the items recommended to a user. Precision evaluates how many of the recommended items are actually relevant to the user's preferences and is calculated as:

$$Precision = \frac{TP}{TP + FP} \quad (19)$$

High precision indicates that the system is making accurate and relevant recommendations.

4.1.2 Recall

Recall is another metric that measures the proportion of relevant items that have been successfully recommended to the user. It evaluates how well the system captures all relevant items that a user might be interested and is calculated as:

$$Recall = \frac{TP}{TP + FN} \quad (20)$$

High recall indicates that the system is effective in recommending a large portion of relevant items.

4.1.3 F-measure

The F-measure is the harmonic mean of Precision and Recall. It is used to provide a balanced evaluation of both metrics. The F-measure is calculated as:

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (21)$$

In recommendation systems, the F-measure can be used to assess the trade-off between making accurate and relevant recommendations (precision) and capturing a large number

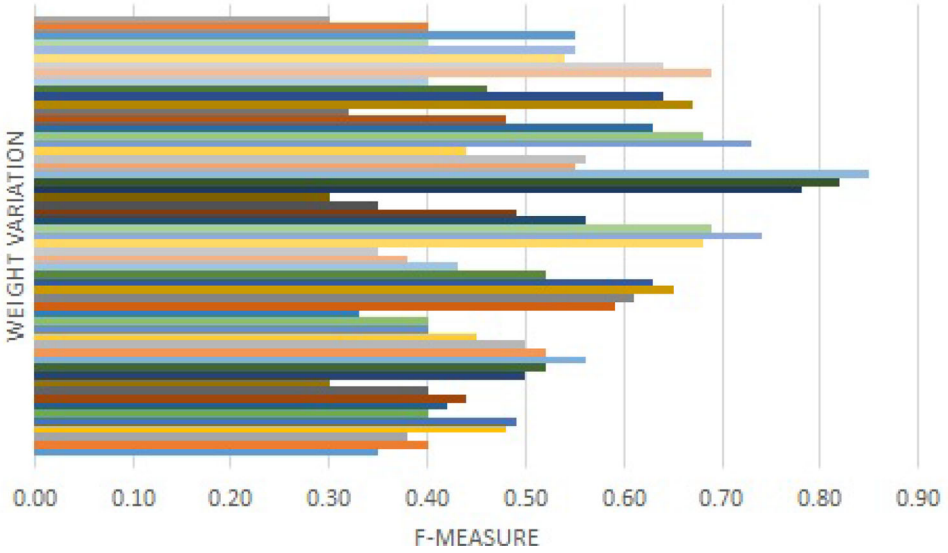


Fig. 12 Values of *F-Measure* with the variation of w_{Upref}, w_{Uloc} and w_{Utrc}

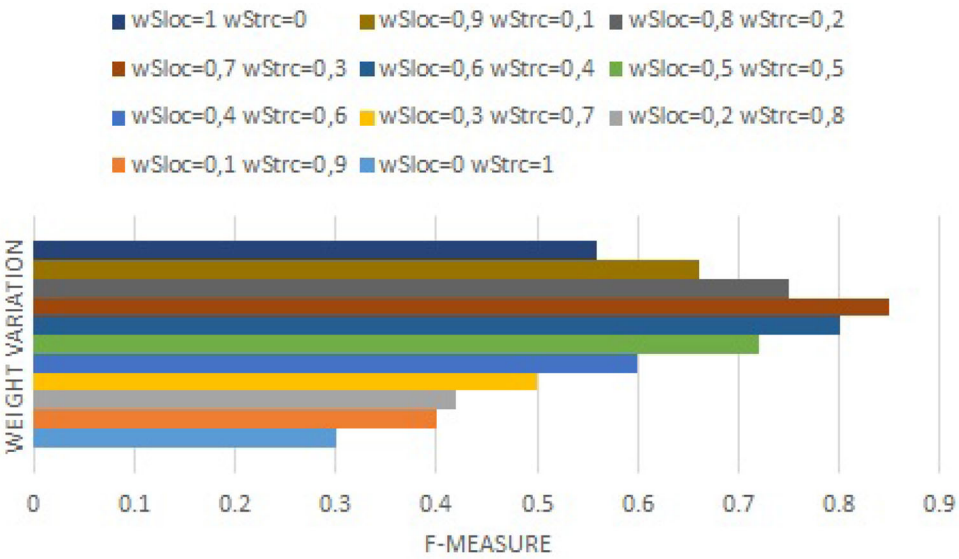


Fig. 13 Values of *F-Measure* with the variation of w_{Sloc} and w_{Strc}

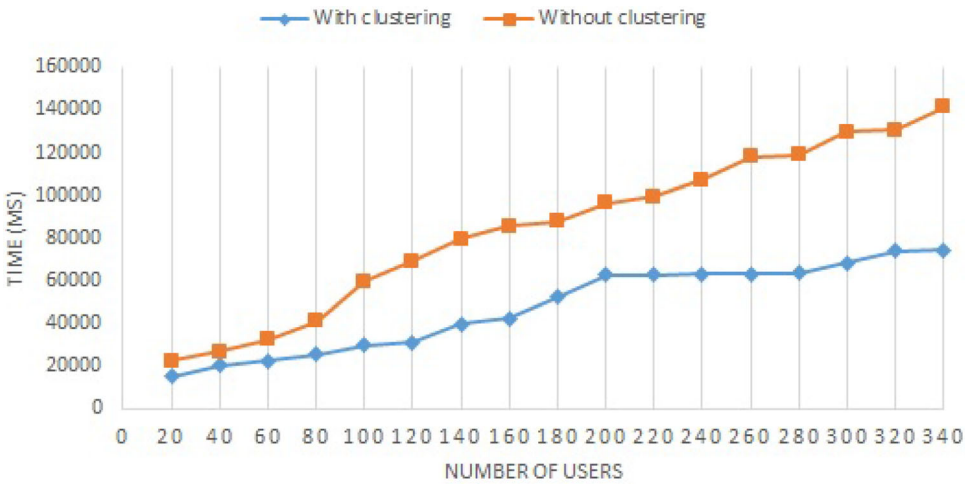


Fig. 14 "HRPCS" with and without clustering

of relevant items (recall). The F-measure ranges from 0 to 1, with higher values indicating better overall performance.

4.1.4 Mean absolute error (MAE)

MAE is used in recommendation systems to assess the accuracy of rating predictions. It measures the average absolute difference between the predicted QoS values and the actual QoS values offered by providers. MAE is calculated as:

$$MAE = \frac{\sum_{i=1}^{n} |\hat{y_{s,i}} - y_{s,i}|}{n} \quad (22)$$

where $\hat{y_{s,i}}$ is the predicted value by the recommendation system for service s and ith QoS parameter, $y_{s,i}$ designates the actual value of the ith QoS parameter given by serviec s, and n is the total number of service-quality pairs (s, i) in the test dataset.

A lower MAE indicates that the recommendations are closer to the actual QoS values, representing a higher accuracy in rating predictions.

4.2 Choice of weight values

The aim of this experiment is to determine the weights values used in Eqs. 1 and 6. Indeed, the similarity between users (Eq. 1) is calculated according to three similarity aspects, namely the QoS preference similarity weighted with w_{Upref}, the Location similarity weighted with w_{Uloc} and the Trace similarity weighted with w_{Utrc}. In order to choose the weights that give the best result, we use the variation of w_{Upref}, w_{Uloc} and w_{Utrc} to determine the best value of *F-Measure*. Figure 12 shows that the best value of *F-Measure* is obtained with $w_{Upref} = 0.5$, $w_{Uloc} = 0.3$ and $w_{Utrc} = 0.2$.

Similarly, service similarity linearly combines two aspects of similarity, namely Location similarity weighted with w_{Sloc} and Trace similarity weighted with w_{Strc} (Eq. 6). From Fig. 13, we get the best result from *F-Measure* when $w_{Sloc} = 0.7$ and $w_{Strc} = 0.3$. This confirms the hypothesis that services in the same location can have similar performance.

4.3 Impact of clustering

To analyze the impact of clustering (Sect. 3.1) on the performance of our "HRPCS" approach, we compare, in this experiment, the recommendation time of "HRPCS" with and without clustering. We calculate the recommendation time of our HRPCS approach based on the number of users. The results show that when the number of users decreases, the recommendation time of "HRPCS" with or without clustering is negligible, but the time of recommendation without clustering increases from 100 users (See Fig. 14). Indeed, the similarities between users are calculated between all users and not within clusters. Therefore, we note that the recommendation-based on clustering is better than the recommendation without clustering, especially when the number of users increases.

4.4 Choice of global QoS threshold ϵ

After calculating the global QoS for each cloud service using Eq. 13, some services, whose global QoS is less than ϵ, will be excluded due to their inconsistency with the QoS preferences requested by the user. In this experiment, we study the impact of the threshold value, ϵ, on the performance of our hybrid recommendation approach, "HRPCS". To do this we use the *Precision*, the *Recall* and the *F-Measure* under different threshold values to

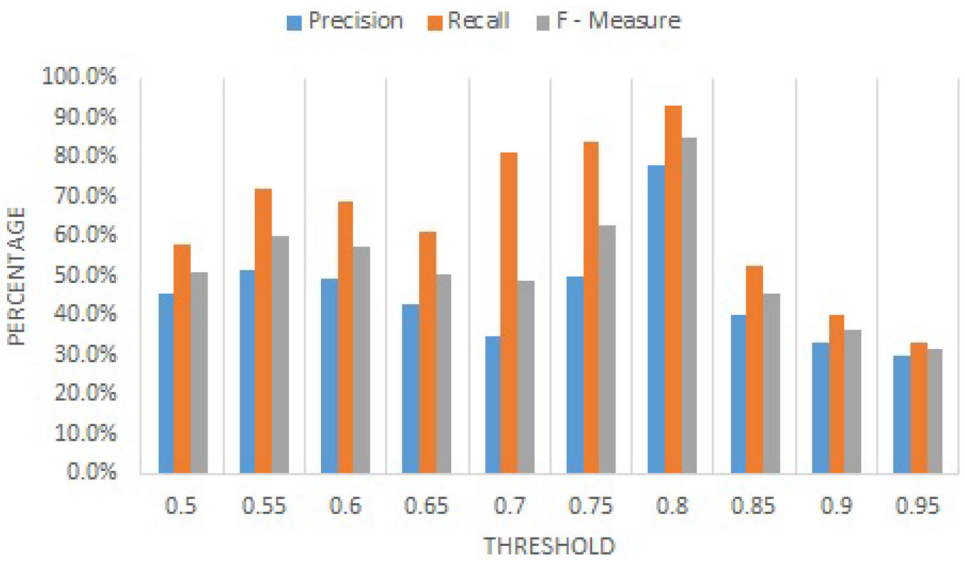


Fig. 15 Determination of the optimal value of the global QoS threshold

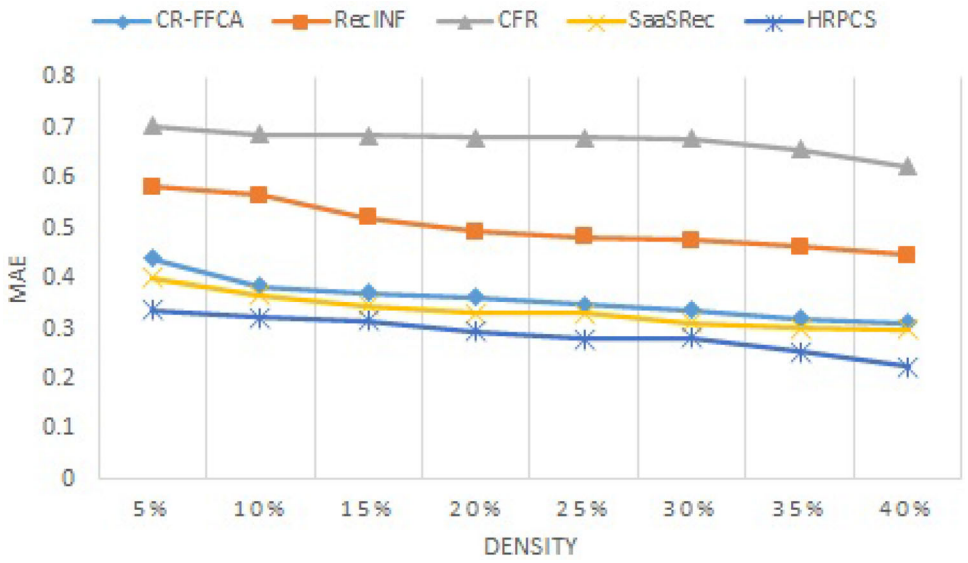


Fig. 16 Recommendation quality evaluation

determine the optimal value. Indeed, based on Fig. 15, we notice that when the threshold value is low ($0.5 \leq \epsilon \leq 0.75$), we get high *Recall* values and low *Precision* values, which means many returned services, but most of these services are irrelevant. Likewise, when we use a high threshold value ($0.85 \leq \epsilon \leq 0.95$), we get low *Recall* values and high *Precision* values, which means that the number of returned services is small and most of these services are relevant. Since we plan to recover as many cloud services as possible, the optimal threshold ϵ will be selected as the value-maximizing both *Precision* and *Recall*. Thus, according to Fig. 15, the optimal global QoS threshold is $\epsilon = 0.8$ when the *Precision*, the *Recall* and the *F- Measure* are respectively equal to 77.8%, 93.3% and 84.8%.

4.5 Comparison with other recommendation approaches

In these experiments, we compare our approach "HRPCS" to CR-FFCA [28], RecINF [47], CFR [29] and SaaSRec [24, 34] in terms of MAE (Mean Absolute Error) values and recommendation time. We believe that the recommendation approaches employed in our evaluation represent the best choices for the comparison as they closely align with our proposal. Moreover, the selected works encompass a diverse range of methodologies and algorithms for service recommendation, including fuzzy formal concept analysis, interval numbers of four parameters, collaborative filtering, and the AHC technique associated with the user similarity measure. Comparing different

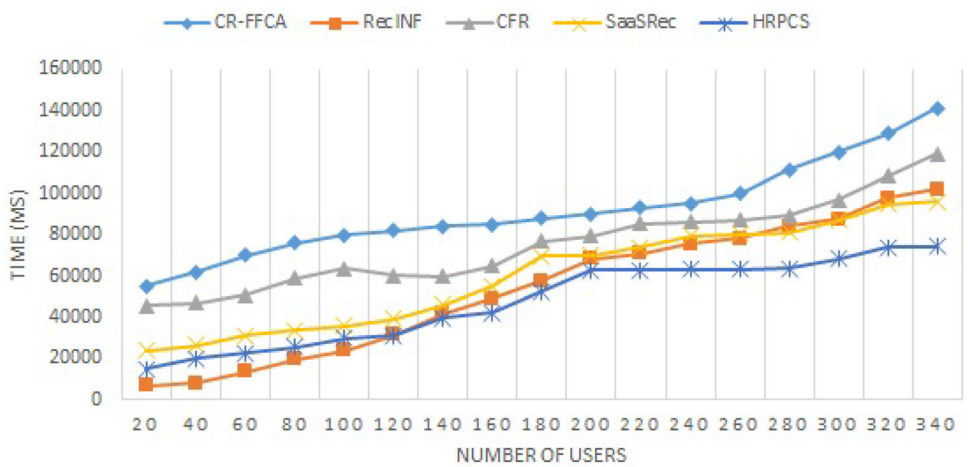


Fig. 17 Recommendation time evaluation

approaches allows us to identify strengths and weaknesses, leading to a more comprehensive understanding of the problem space. Finally, since the selected works have publicly available results and datasets, it becomes easier to compare them with our proposed approach.

Since the service-quality matrix is generally sparse, we vary the density of the service-quality matrix from 10% to 40% with a step value of 10% to study the prediction accuracy of our approach under different matrix density conditions. According to the results shown in Fig. 16, the MAE values of all approaches decrease with increasing matrix density, indicating that the prediction accuracy can be improved by providing more information on QoS. We note that our approach achieves the smaller MAE values indicating better prediction accuracy. This result occurs because our approach is protected against malicious user attack by neglecting user feedback and focusing on predicting missing QoS values taking into account information from similar services to make the predictions. Concretely, compared to the SaaSRec approach, our approach achieves an average improvement of 14.2%. The MAE values of CFR are the highest because this approach uses unreliable data provided by untrusted users to make predictions. The other approaches obtained better accuracy because they reduced the impact of data provided by untrusted users. The experimental results indicate that the treatment of unreliable data is essential for a robust prediction approach.

The recommendation times for the different approaches are shown in Fig. 17. We notice that the recommendation time of all approaches increases when the users' number increases. The CR-FFCA approach requires considerable recommendation time to recommend services. The CFR approach takes less time than CR-FFCA. The recommendation time spent by SaaSRec is close to that of our approach. Indeed, SaaSRec groups similar users into communities, and similarities are calculated only within these communities. The recommendation time for RecINF is acceptable when the users' number is small. However, it quickly becomes high when the users' number exceeds 100. This is because the RecINF approach is not based on clustering. In general, the recommendation time of our approach is faster than that of the four recommendation approaches. This observation indicates that by combining user and service clustering with recommendation algorithms, our approach can achieve better recommendation time.

4.6 Complexity analysis

Computational complexity is a term used in computer science to describe the amount of resources, specifically time, required to run an algorithm. In "HRPCS", the evolutionary search largely determines the computational complexity. The "Clustering Module" is the first stage of its evolutionary search. The AHC method, on which this module is based, has an $O(n^3)$ complexity. The three phases of the "Hybrid Recommendation Module" are mostly responsible for determining its complexity. "Recovery of used services", "Functional and non-functional filtering", and "Filtering based on QoS preferences" have complexity factors of $O(n)$, $O(n)$, and $O(n^2)$, respectively. Consequently, the overall computational complexity of the "Hybrid Recommendation Module" is $O(n^2)$. The third stage of the evolutionary search is the "Ranking and Diversification Module". The complexity of the algorithms for service ranking and service diversification is $O(n)$ and $O(n)$, respectively. As a result, the complexity of the third stage is $O(n)$.

5 Conclusion

In this paper, we have presented an approach to personalized cloud service recommendation. This approach meets a principal objective that is to improve user satisfaction by offering cloud services in line with their requirements but sufficiently diversified to arouse new interests in them. We presented our hybrid recommendation architecture that delivers a personalized list of services based on the user's needs (functional and non-functional) and the user's QoS preferences. Our hybrid approach consists of three recommendation techniques, namely demographic, knowledge-based, and collaborative, each of these techniques is activated at a specific recommendation phase. The results of the hybrid recommendation are then ranked according to a score function integrating two criteria, namely the price and the credibility of the services. Finally, to offer to the user a list of recommendations without redundancy, our approach that is based on diversity builds a diversified list of cloud services.

In the future, we plan to add a monitoring module to provide the QoS parameters values in real-time. Indeed, by using a monitoring module, the approach can detect violations and issue triggers to take the necessary measures to execute the corresponding SLA compensation clauses. Additionally, the suggested approach can be modified for use at run-time, for instance, when a selected cloud service needs to be changed because it is no longer functioning properly or when a new, more effective service becomes available.

Author contributions All authors reviewed the manuscript.

Declarations

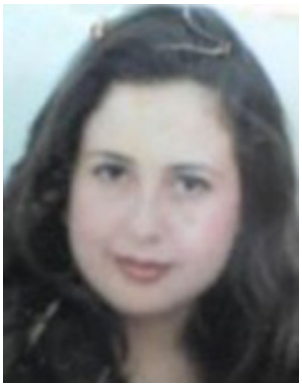
Competing interests The authors declare no competing interests.

References

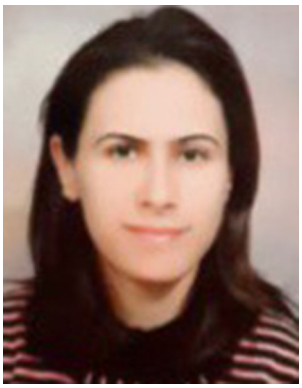
1. Mell, P., Grance, T., et al.: The NIST Definition of Cloud Computing. National Institute of Standards and Technology, Gaithersburg (2011)
2. Nagarajan, R., Selvamuthukumaran, S., Thirunavukarasu, R.: A fuzzy logic based trust evaluation model for the selection of cloud services. In: International Conference on Computer Communication and Informatics (ICCCI), pp. 1–5 (2017). IEEE
3. Esposito, C., Ficco, M., Palmieri, F., Castiglione, A.: Smart cloud storage service selection based on fuzzy logic, theory of evidence and game theory. IEEE Trans. Comput. **65**(8), 2348–2362 (2015)
4. Nabli, H., Ben Djemaa, R., Ben Amor, I.A.: Efficient cloud service discovery approach based on lda topic modeling. J. Syst. Softw. **146**, 233–248 (2018)
5. Ben Djemaa, R., Nabli, H., Ben Amor, I.A.: Enhanced semantic similarity measure based on two-level retrieval model. Concurr. Comput. **31**(15), 5135 (2019)
6. Abbas, G., Mehmood, A., Lloret, J., Raza, M.S., Ibrahim, M.: Fipa-based reference architecture for efficient discovery and selection of appropriate cloud service using cloud ontology. Int. J. Commun Syst **33**(14), 4504 (2020)
7. Hajlaoui, J.E., Omri, M.N., Benslimane, D.: A QoS-aware approach for discovering and selecting configurable iaas cloud services. Comput. Syst. Sci. Eng. **32**(4), 460–467 (2017)
8. Eisa, M., Younas, M., Basu, K., Awan, I.: Modelling and simulation of QoS-aware service selection in cloud computing. Simul. Model. Pract. Theory **103**, 102108 (2020)
9. Youssef, A.E.: An integrated mcdm approach for cloud service selection based on topsis and bwm. IEEE Access **8**, 71851–71865 (2020)
10. Sindhu, K., Guruprasad, H.S.: Computational offloading framework using caching and cloud service selection in mobile cloud computing. Int. J. Adv. Intell. Paradig. **21**(3–4), 189–210 (2022)
11. Tiwari, R.K., Kumar, R.: G-topsis: a cloud service selection framework using gaussian topsis for rank reversal problem. J. Supercomput. **77**, 523–562 (2021)
12. Kumar, R.R., Mishra, S., Kumar, C.: Prioritizing the solution of cloud service selection using integrated mcdm methods under fuzzy environment. J. Supercomput. **73**, 4652–4682 (2017)
13. Gabi, D., Ismail, A.S., Zainal, A., Zakaria, Z., Abraham, A., Dankolo, N.M.: Cloud customers service selection scheme based on improved conventional cat swarm optimization. Neural Comput. Appl. **32**, 14817–14838 (2020)
14. Mohamed, A.M., Abdelsalam, H.M.: A multicriteria optimization model for cloud service provider selection in multicloud environments. Software **50**(6), 925–947 (2020)
15. Nabli, H., Ben Djemaa, R., Ben Amor, I.A.: Description, discovery, and recommendation of cloud services: a survey. SOCA **16**(3), 147–166 (2022)
16. Lops, P., De Gemmis, M., Semeraro, G.: Content-Based Recommender Systems: State of the Art and Trends. Recommender Systems Handbook, pp. 73–105 (2011)
17. Schafer, J.B., Frankowski, D., Herlocker, J., Sen, S.: Collaborative filtering recommender systems. In: The Adaptive Web: Methods and Strategies of Web Personalization, pp. 291–324 (2007). Springer
18. Yao, L., Sheng, Q.Z., Ngu, A.H., Yu, J., Segev, A.: Unified collaborative and content-based web service recommendation. IEEE Trans. Serv. Comput. **8**(3), 453–466 (2014)
19. Gao, X., Yu, C.: A fuzzy-based recommendation system for cloud accounting service. In: 13th International Conference on Service Systems and Service Management (ICSSSM), pp. 1–6 (2016). IEEE
20. Jain, G., Mahara, T., Sharma, S.C., Sangaiah, A.K.: A cognitive similarity-based measure to enhance the performance of collaborative filtering-based recommendation system. IEEE Trans. Comput. Soc. Syst. **9**(6), 1785–1793 (2022)
21. Najafabadi, M.K., Mohamed, A., Onn, C.W.: An impact of time and item influencer in collaborative filtering recommendations using graph-based model. Inf. Process. Manag. **56**(3), 526–540 (2019)
22. Burke, R., O'Mahony, M.P., Hurley, N.J.: Robust collaborative recommendation. In: Recommender Systems Handbook, pp. 961–995 (2015). Springer
23. Soltani, S., Elgazzar, K., Martin, P.: Quaram service recommender: a platform for iaas service selection. In: 2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC), pp. 422–425 (2016). IEEE
24. Afify, Y.M., Moawad, I.F., Badr, N.L., Tolba, M.F.: A personalized recommender system for saas services. Concurr. Comput. **29**(4), 3877 (2017)
25. Balaji, S., Rajkumar, K.: A personalized cloud service recommendation system using collaborative filtering. Int. J. Pure Appl. Math. **119**(12), 14173–14180 (2018)
26. Ding, S., Wang, Z., Wu, D., Olson, D.L.: Utilizing customer satisfaction in ranking prediction for personalized cloud service selection. Decis. Support Syst. **93**, 1–10 (2017)
27. Ma, H., Hu, Z., Li, K., Zhu, H.: Variation-aware cloud service selection via collaborative QoS prediction. IEEE Trans. Serv. Comput. (2019). https://doi.org/10.1109/TSC.2019.2895784
28. Mezni, H., Abdeljaoued, T.: A cloud services recommendation system based on fuzzy formal concept analysis. Data Knowl. Eng. **116**, 100–123 (2018)
29. Wang, F.F., Chen, F.Z., Li, M.Q.: A collaborative filtering method for cloud service recommendation via exploring usage history. In: Proceeding of the 24th International Conference on Industrial Engineering and Engineering Management 2018, pp. 716–725 (2019). Springer
30. Djiroun, R., Guessoum, M.A., Boukhalfa, K., Benkhelifa, E.: A novel cloud services recommendation system based on automatic learning techniques. In: International Conference on New Trends in Computing Sciences (ICTCS), pp. 42–49 (2017). IEEE
31. Zheng, X., Da Xu, L., Chai, S.: QoS recommendation in cloud services. IEEE Access **5**, 5171–5177 (2017)
32. Nagarajan, R., Thirunavukarasu, R.: A service context-aware QoS prediction and recommendation of cloud infrastructure services. Arab. J. Sci. Eng. **45**(4), 2929–2943 (2020)
33. Ngaffo, A.N., El Ayeb, W., Choukair, Z.: Service recommendation driven by a matrix factorization model and time series forecasting. Appl. Intell. 1–16 (2021)
34. Afify, Y.M., Moawad, I.F., Badr, N.L., Tolba, M.F.: Enhanced similarity measure for personalized cloud services recommendation. Concurr. Comput **29**(8), 4020 (2017)
35. Tang, M., Jiang, Y., Liu, J., Liu, X.: Location-aware collaborative filtering for QoS-based service recommendation. In: 2012 IEEE 19th International Conference on Web Services, pp. 202–209 (2012). IEEE
36. Lo, W., Yin, J., Li, Y., Wu, Z.: Efficient web service QoS prediction using local neighborhood matrix factorization. Eng. Appl. Artif. Intell. **38**, 14–23 (2015)
37. Zheng, Z., Ma, H., Lyu, M.R., King, I.: Wsrec: a collaborative filtering based web service recommender system. In: 2009 IEEE

International Conference on Web Services, pp. 437–444 (2009). IEEE
38. Nabli, H., Ben Djemaa, R., Ben Amor, I.A.: Linked usdl extension for cloud services description. In: International Conference on Web Engineering, pp. 359–373 (2019). Springer
39. Nabli, H., Ben Djemaa, R., Ben Amor, I.A.: Cloud services description ontology used for service selection. Service Oriented Computing and Applications, 1–14 (2021)
40. Sha, L., Shaozhong, G., Xin, C., Mingjing, L.: A QoS based web service selection model. In: 2009 International Forum on Information Technology and Applications, vol. 3, pp. 353–356 (2009). IEEE
41. Kang, G., Tang, M., Liu, J., Liu, X., Cao, B.: Diversifying web service recommendation results via exploring service usage history. IEEE Trans. Serv. Comput. **9**(4), 566–579 (2015)
42. Alves, M.J., Costa, J.P.: Graphical exploration of the weight space in three-objective mixed integer linear programs. Eur. J. Oper. Res. **248**(1), 72–83 (2016)
43. Kunaver, M., Požrl, T.: Diversity in recommender systems-a survey. Knowl.-Based Syst. **123**, 154–162 (2017)
44. Ziegler, C.N., McNee, S.M., Konstan, J.A., Lausen, G.: Improving recommendation lists through topic diversification. In: Proceedings of the 14th International Conference on World Wide Web, pp. 22–32 (2005)
45. Santos, R.L., Macdonald, C., Ounis, I.: Search result diversification. Found. Trends Inf. Retr. **9**(1), 1–90 (2015)
46. Ricci, F., Rokach, L., Shapira, B., Kantor, P.B.: Recommender Systems Handbook. Springer, Berlin (2010)
47. Ma, H., Hu, Z.: Recommend trustworthy services using interval numbers of four parameters via cloud model for potential users. Front. Comput. Sci. **9**(6), 887–903 (2015)

Hajer Nabli proudly earned her Master's degree in Computer Networks from the Higher Institute of Computer Science and Communication Technologies of Hammam Sousse, part of Sousse University, Tunisia, in 2015. This educational milestone laid the foundation for her future pursuits and ignited her fascination with cutting-edge technologies. Building upon her master's studies, she embarked on a remarkable journey that culminated in the acquisition of a Ph.D. in Computer Science from the esteemed Faculty of Economics and Management of Sfax at Sfax University, Tunisia. In 2021, she successfully defended her doctoral thesis, a testament to years of relentless research, innovation, and scholarly dedication. Throughout her academic endeavors, she has had the privilege of being an active member of the MIRACL laboratory at Sfax University. Her research areas span a diverse range of topics, including Cloud Computing, Cloud Service Recommendation, the Internet of Things (IoT), Artificial Intelligence (AI), Machine Learning, and Blockchain.



Raoudha Ben Djemaa received her Master degree in Information system and New Technology from Sfax University, Tunisia, in 2002. She obtained her PhD in computer science from Sfax University, Tunisia, in Avril 2009. She is actually an Associate Professor in the Higher Institute of Computer Science and Communication Technologies of Hammam Sousse in Tunisia (ISITCom). She is also a member of the laboratory MIRACL (Multimedia, Information Systems and Advanced Computing) of Sfax University, Tunisia. Her research interests include software engineering, methodologies and approaches for adaptive web applications, Development and approaches for adaptive web services and finally IHM adaptation. She has participated in several National and International conferences.



Ikram Amous Ben Amor received her Master's degree in Data Processing from Paul Sabatier University (Toulouse III, France), France, in 1999. She obtained her Ph.D. in computer science from the Paul Sabatier University, in December 2002. She is currently a Professor at the National School of Electronics and Telecommunications of Sfax in Tunisia (ENET'Com). She is also a member of MIRACL laboratory of Sfax University, Tunisia. Her research interests include social network and user profile analysis, adaptive semi-structured document, etc. She has participated in several program committees of national and international conferences.