The Journal of Systems and Software 93 (2014) 217–228

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

A trustworthy QoS-based collaborative filtering approach for web service discovery

Szu-Yin Lin [a], Chin-Hui Lai [a,*], Chih-Heng Wu [b], Chi-Chun Lo [b]

[a] Department of Information Management, Chung Yuan Christian University, Taoyuan County, Taiwan
[b] Institute of Information Management, National Chiao Tung University, Hsin-Chu, Taiwan

ARTICLE INFO

Article history:
Received 13 September 2012
Received in revised form
19 November 2013
Accepted 23 January 2014
Available online 3 February 2014

Keywords:
Service selection
Service discovery
Quality of Service (QoS)
Collaborative filtering

ABSTRACT

Many network services which process a large quantity of data and knowledge are available in the distributed network environment, and provide applications to users based on Service-Oriented Architecture (SOA) and Web services technology. Therefore, a useful web service discovery approach for data and knowledge discovery process in the complex network environment is a very significant issue. Using the traditional keyword-based search method, users find it difficult to choose the best web services from those with similar functionalities. In addition, in an untrustworthy real world environment, the QoS-based service discovery approach cannot verify the correctness of the web services' Quality of Service (QoS) values, since such values guaranteed by a service provider are different from the real ones. This work proposes a trustworthy two-phase web service discovery mechanism based on QoS and collaborative filtering, which discovers and recommends the needed web services effectively for users in the distributed environment, and also solves the problem of services with incorrect QoS information. In the experiment, the theoretical analysis and simulation experiment results show that the proposed method can accurately recommend the needed services to users, and improve the recommendation quality.

1. Introduction

Due to recent advances in ubiquitous and distributed computing, many distributed network services which process a large quantity of data and knowledge are available in the cloud computing network environment, and provide useful data and applications to users based on Service-Oriented Architecture (SOA) and web services technology. Because some service providers may provide services with similar functionalities, data, and knowledge, it is difficult for users to find relevant and the best services from many different types of services. Therefore, recommending an appropriate web service to fulfill a user's requirements from a large number of services is a great challenge.

In SOA, many studies have focused on the service discovery problem (González-Valenzuela et al., 2011; Koloniari and Pitoura, 2004). The traditional method of web service discovery is Universal Description Discovery and Integration (UDDI), which works as an intermediary between service providers and users. In UDDI systems, service providers can publish meta-information of services for locating web services, while users can send their requirements as queries to search for required services. Although the OASIS UDDI Specification Technical Committee voted to complete UDDI's final standard version in late 2007 and close the project of UDDI, UDDI systems are still most commonly found inside companies or inter-companies, where they are used to dynamically bind client systems to implementations. Because the number of services is increasing constantly, methods of service selection and service discovery are necessary for improving search efficiency. However, the methods of service discovery provided by UDDI can only satisfy users' functional requirements (Xu et al., 2007; Yan and Piao, 2009), and ignore users' non-functional requirements. Generally, functional requirements define what a system is supposed to do, whereas non-functional requirements define how a system is supposed to be. In recent years, non-functional requirements are also called quality attributes. Quality of Service (QoS) is usually employed to describe non-functional attributes (e.g., response time, availability, throughput, price, cost, reliability, etc.) of web services (Raj and Sasipraba, 2010). The QoS values are user-dependent because users may give different values based on their demands. Consequently, QoS can be applied in solving the problem of service selection when choosing between services which have similar functionalities (Canfora et al., 2008; Ko et al., 2008).

Providing the QoS information of web services could help users to select an appropriate service from many with the same functionality. However, there are some problems for service discovery in QoS. Because some QoS information in UDDI may be untrustworthy

* Corresponding author. Tel.: +886 3 2655413; fax: +886 3 2655499.
E-mail addresses: chlai@cycu.edu.tw, chinhui.lai@gmail.com (C.-H. Lai).

0164-1212/$ – see front matter © 2014 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2014.01.036

(such as outdated information of services, or unavailable services), UDDI-discovered web services approach may not meet users' non-functional requirements. Essentially, traditional QoS-based query methods cannot verify the correctness of QoS information. Some methods, i.e., Web Services Level Agreements (WSLA) (Keller and Ludwig, 2003), reputation-enhanced methods (Ali et al., 2006; Maximilien and Singh, 2005; Wishart et al., 2005; Xu et al., 2007), and collaborative filtering methods (Manikrao and Prabhakar, 2005; Zheng et al., 2009) are proposed. WSLA is a standard for service level agreement compliance monitoring of web services. It allows users to specify the QoS performance metrics associated with a web service application (such as service selection or discovery), desired performance targets, and actions that should be performed when QoS performance is not met. The reputation-enhanced methods aggregate the QoS ratings of services as web service reputation to reflect the overall QoS for ranking the services during the service discovery process. Such service reputation also provides a general estimate for the reliability of service providers. Considering the web service reputation may increase the probability of finding the reliable services.

Moreover, collaborative filtering methods are widely used in many application domains. They are generally classified into user-based and item-based collaborative filtering methods based on similar users and similar items respectively to predict the values of items. For the web service discovery, the user-based collaborative filtering methods employ historical QoS information obtained from users who have similar QoS experience on the use of web services, and then predict the QoS performance of a web service for an active user. Similarly, the item-based methods use the similarity between the web services instead of the service users to predict a QoS value of a web service. The mentioned approaches have been proposed to solve the QoS incorrectness problem in service discovery, but they still cannot completely solve the untrustworthy QoS problem. Because of WSLA, it is impossible for users to sign the WSLA contract of every service provider when they query through a UDDI registry center. For reputation-enhanced methods, the reputation values of services may be incorrect because of malicious user feedback, malicious attacks, or inaccurate ratings given by users. Moreover, traditional collaborative filtering methods not only have the same problems as reputation-enhanced methods, but may also be unable to recommend the needed web services without users' rating data.

Therefore, this study aims to solve the following problems for service discovery in QoS: (1) QoS-based query methods cannot verify the correctness of the QoS information. (2) When users' ratings of web services are collected in query methods, it is necessary to deal with the problems of malicious users and malicious attacks. (3) Because feedback scores from users for web services are subjective, inaccuracies or errors in computing the reputation values of services may occur. (4) Traditional collaborative filtering approaches cannot recommend services corresponding to users' QoS requirements.

This paper proposes a trustworthy two-phase web service discovery mechanism based on collaborative filtering and QoS information. The first phase is used to filter out services with incorrect QoS values, while the second phase is employed to recommend services to users according to their non-functional requirements, which describe the correct QoS information for service consumers. Therefore, the research objectives in this paper are: (1) to recommend web services corresponding to services' functionalities and users' QoS requirements, (2) to filter out web services with incorrect QoS parameters, and (3) to propose a trustworthy platform for service discovery.

The rest of this paper is organized as follows: Section 2 describes the related works, including web services architecture, QoS storage for web services, QoS-based service discovery methods, and recommendation methods. In Section 3, we propose a trustworthy two-phase web service discovery mechanism. The details of our method are clearly illustrated. Section 4 shows the implementation of our proposed method for the experiment, and the evaluation of the experimental results. Finally, our conclusion and future works are discussed in Section 5.

2. Related works

In this section, we will discuss the related works on the basic framework and development of web services, QoS storage for web services, QoS-based web service discovery methods, and collaborative filtering for web service recommendations.

2.1. Web services architecture

The World Wide Web Consortium (W3C) describes web services as a software system to support interoperable machine-to-machine interaction over a network (Haas and Brown, 2004). For the B2C environment, the development of web services focuses on individual users who can only use the limited functions provided by a single computer. To create an operation process between the services within an enterprise and its partners in a B2B environment, the concept of Service-Oriented Architecture (SOA) is proposed. Thus, software applications, not only services, can communicate with each other based on SOA. These applications can exchange and integrate information based on SOA and XML-based web service techniques. W3C (World Wide Web Consortium) defines web services as an application that can use XML to describe a request and utilize URI (Uniform Resource Identifier) to identify the interface of the application and binding methods. Thus, the message between applications is transferred over the Internet using XML and standard Internet techniques. That is, the web service is regarded as a software component, and applies the HTTP protocol and open data format, e.g., XML, WSDL and SOAP, to communicate with other applications. Because web services technology is based on the Internet, it provides an effective means of communication. Even though it can be implemented by using different programming languages and platforms, it is possible to resolve the integration problems and difficulties in distributed systems. There are 3 kinds of roles in Service-Oriented Architecture, including service provider, service requester, and UDDI registry (Papazoglou, 2003). The service mainly provides the development of web services, descriptions of creating the web service, and the service registration in UDDI. UDDI, which is a mediator between service provider and service requester, receives registration requests from a service provider, and deals with queries from a service requester. Thus, a service requester queries and searches a specific service in UDDI when he has some demands. The process for obtaining a web service is as follows.

(1) *Registry*: A service provider provides the needed service information to UDDI.
(2) *Publish*: UDDI accepts the registration of a service by a service provider and informs other UDDIs.
(3) *Find*: A service requester sends a query requirement to UDDI.
(4) *Response of a service query*: UDDI replies with a query result to the service requester, and retrieves the service WSDL.
(5) *Invoke*: Based on the description of the WSDL, the service requester sends a request to a service provider.
(6) *Bind*: The service provider replies with the result of the request.

According to the above process, the "Find" step is important in service discovery to help service requesters discover the services they need. In the following sections, this paper will investigate and discuss the details of the problem associated with service discovery.

2.2. QoS storage for web services

According to the definition given by Menasce (Menasce, 2002), Quality of Service (QoS) is composed of several service-related attributes. These attributes are non-functional attributes, and are becoming increasingly more important in finding services. The most widely used techniques for discovering web services, e.g., WSDL or UDDI, only consider users' functional requirements. Users may have difficulty in finding the best service from a large number of services with the same functions. Thus, QoS becomes an important indicator in finding services (Canfora et al., 2008; Ko et al., 2008; Mokhtar et al., 2008; Yang et al., 2012). Generally, the QoS information may include the following attributes (Huang et al., 2009; Menasce, 2002; Raj and Sasipraba, 2010):

- Price is the cost of a request to a service, and it is usually charged per transaction.
- Availability is the probability that a service can respond to a request within a period of time.
- Accessibility is the probability that a system can work normally, and process requests without any delay.
- Response time is the time taken for a user to receive a response after his/her request.
- Throughput is the number of requests that a service can process within a given period of time.

In UDDI Version 3, the query methods provided for users are only able to meet the functional requirements by description in UDDI for web service selection and discovery, while service providers cannot register their QoS with UDDI. In order to solve the problem of QoS requirements, some studies have focused on embedding QoS information within a message to enhance inquiry operations. For example, Ali's (ShaikhAli et al., 2003) UDDIe, which provides an application program interface (API) for users to set QoS information via *propertyBag*. However, this method can only be implemented in the G-QoSM framework of UDDIe, and provides limited QoS information. Martin-Diaz et al. (2003) proposes a quality requirement language (QRL) that is an XML-based language, to be used in web services and distributed systems for specifying quality requirements. However, it fails to define the relationship between QoS information and service interface standards, e.g., WSDL. Besides, QRL does not provide enough information to illustrate how to manage and maintain the QoS information in web services.

The most common method of publishing QoS information in UDDI is *tModel* (Ran, 2003; Xu et al., 2007). *tModel* in UDDI is used to describe the technical fingerprint of a web service, which consists of format, protocol, input and output parameters, business logic, and so on. It is typically used to specify the details in the description of a web service implementation. A web service entry in UDDI can refer to multiple *tModel*s, registered in UDDI, and includes multiple property information. In UDDI, each property is represented by a *keyedReference*, which is a general-purpose structure for a name-value pair in the *tModel*. The name of a QoS attribute is specified by a *keyName*, and its value is specified by a *keyValue*. Thus, *tModel* is based on the existing structures in UDDI. It is not necessary to develop additional standards.

2.3. QoS-based service discovery methods

Finding an appropriate and best service from a group of services with the same functionality is a service selection challenge (Liangzhao et al., 2004; Maximilien and Singh, 2004; Mukhopadhyay and Chougule, 2012; Nair and Gopalakrishna, 2010). Therefore, besides the functionalities of web services, non-functional attributes (QoS) are also important in service discovery. In order to meet users' functional and non-functional requirements

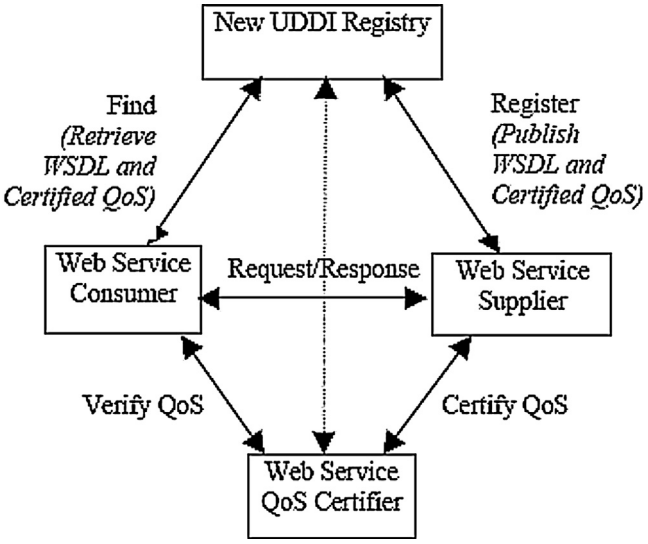

Fig. 1. A new service registration and discovery model (Ran, 2003).

in service discovery, many QoS-based methods have been proposed (Harshavardhanan et al., 2012; Torres et al., 2011). Gouscos (Gouscos et al., 2003) classifies QoS attributes as a static and a dynamic group. For example, price, promised response time, and probability of failure are static attributes stored in UDDI. On the other hand, actual response time, and failure rate are stored in WSDL, or provided by an information broker. This method is quite easy and straightforward, but cannot solve the problem caused by obsolete and out-of-date QoS information. Ran (2003) proposes an extended UDDI model which involves a new role, namely a web service QoS Certifier, in a traditional SOA model. It consists of three roles (service providers, service requesters, and UDDI registry), as shown in Fig. 1. This verifies whether the quality of a service is the same as the service provider promised before registering the service in UDDI. Thus, a service consumer can first issue a query to the web service QoS Certifier to verify the QoS of a target service. However, this does not provide a reliable algorithm to validate the credibility of claimed QoS. It is unable to assure the correctness of real QoS when web services are changed or updated in the future.

Huang et al. (2009) proposes a three-stage service selection scheme based on different types of QoS, as shown in Fig. 2. In a text-based QoS matchmaking stage, keyword search and service category search are provided by UDDI. There are two scenarios in a stage of numeric-based QoS matchmaking: single QoS-based service discovery, which selects the service with the best QoS attributes for users, and QoS-based Optimization, which selects the service with the best performance in an entire workflow.

Al-Masri and Mahmoud (2007) proposes a web Service Repository Builder (WSRB) framework. In this framework, a Web Service Crawler Engine (WSCE) sends multiple queries to several UDDI registry centers, according to users' requests, and then gets all the QoS results. The QoS of services with the same functionality will be stored in a matrix, and all QoS attributes are then normalized. The ranking scores for each service will be computed by a weighted sum of the value of QoS attributes, where the weighting for each attribute is determined by a user according to his/her preferences.

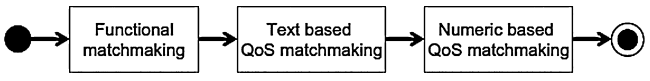

Fig. 2. Three-stage service selection model (Huang et al., 2009).

Finally, services with high ranking scores are recommended to the user. Though this study proposes a novel approach for service selection, the issue of QoS correctness in UDDI is not discussed and verified. In other words, the problem of trustworthy QoS information in QoS-based service selection methods is still not resolved.

2.4. Collaborative filtering for service discovery

Since some studies have focused on services based on analyzing users' usage behavior, we will briefly introduce collaborative filtering, which is a widely used recommendation method in various application domains, and then discuss the query methods of web services using the collaborative filtering algorithm.

2.4.1. Collaborative filtering (CF)

Collaborative filtering (CF) mainly utilizes a collection of users with similar interests (i.e., similar user profiles) to predict interesting information (items) for target users (Herlocker et al., 2000; Wang et al., 2008). There are three main algorithms for CF: user-based, item-based, and hybrid of collaborative filtering. The user-based collaborative filtering algorithm (Wang et al., 2008) predicts a target user's interest in an item based on the ratings from other similar users. Notably, the rating is a quantitative measurement of a user' preference to an item. Thus, items with high predicted scores are recommended to the target user. A user-item matrix where each entry is a rating $x_{a,b}$ of item i_b given by user u_a can be used to estimate the similarities between the target user and all other users. These similarities are then arranged from highest to lowest, and the predicted rating for the target user is derived from the ratings given by similar users.

In contrast to the user-based approach, the item-based collaborative filtering algorithm (Konstan et al., 1997; Sarwar et al., 2001) calculates the similarities between items, rather than the similarities between users. The item similarities are then used to predict ratings of items for users indirectly, by finding items that are similar to other items the users have previously accessed. Finally, the items with high predicted scores are recommended to the target users. The item-based approach has a user-item matrix where each entry is a rating $x_{a,b}$ of item i_b given by user u_a can be used to estimate the similarities between items. These similarities are then arranged from highest to lowest, and the predicted rating of a target item is derived from the ratings of similar items.

However, these methods have a common shortcoming in that they only take the user or item perspective into consideration. For the user-based approach, if only a few users are similar to a target user, the prediction is unreliable and likely to be incorrect. Similarly, the item-based approach may make inaccurate predictions for items if only a few items are similar to the target items. To solve such problems, Wang et al. (2006) proposes a hybrid collaborative filtering algorithm to take both the user and item perspectives into account. Their approach first measures users' similarities and item similarities, and then integrates the similarities to make predictions.

2.4.2. Query methods using collaborative filtering

To date, only a few research works have applied collaborative filtering to enhance web service discovery (Comuzzi and Pernici, 2009; Fernandez et al., 2008; Khapre and Chandramohan, 2011; Li et al., 2012). For example, Birukou et al. (2007) records user queries and requests of service, and analyzes the relations between a query string and the web service in order to identify request-solution pairs, which match user requests with web services. Manikrao and Prabhakar (2005) presents a web service selection framework that combines the item-based collaborative filtering algorithm and semantic matching of service requirement. This helps users query web services which can meet users' QoS requirements, and solves the web service selection problems. However, predictions based on such a framework are inaccurate when only a few items are similar to the target item. Therefore, Zheng et al. (2009) proposes a hybrid collaborative filtering algorithm, which takes both the similarity between users and the similarity between web services into account in order to query web services. The approach predicts the QoS rating of a specific web service for a target user by calculating the scores from the user-based and item-based collaborative filtering methods, respectively. These two scores are then linearly combined by using a weight, which indicates the relative importance of these two kinds of collaborative filtering methods, to derive the predicted score. Thus, web services with high predicted scores are recommended. This approach takes both perspectives of user and web service into general consideration, so that the predicted scores will be more accurate. However, these two approaches do not consider malicious users and malicious attacks. Moreover, user ratings are subjective, which might lead to inaccurate predictions.

In addition, there is a recommendation method that recommends web services based on the analysis of users' behavior instead of web service ratings. For example, Birukou et al. (2007) records user queries and requests of service, and analyzes the relations between a query string and the web service in order to identify request-solution pairs, which match user requests with web services. The similarities between users' requests are then evaluated, and collaborative filtering is used to recommend web services based on these similarities. Kerrigan (2006) transforms the item-user matrix in the collaborative filtering algorithm into a service-goal matrix. Notably, the service represents web services, while the goal represents the semantics in users' requirements. The service-goal matrix is used to record the frequency counts that indicate how many web services match the goals. Based on the frequency, the relations between web services and goals are analyzed, and web services with high frequencies are recommended. These approaches do not collect user ratings of web services, so they do not have to take malicious users and malicious attacks into account. However, they cannot recommend web services according to the users' QoS requirements.

3. A trustworthy two-phase web service discovery mechanism

This study proposes a trustworthy two-phase web service discovery mechanism based on collaborative filtering and QoS. The architecture of our proposed method is shown in Fig. 3. In the first phase of proposed architecture, the observer agents will collect records of user behavior, including querying and invoking web services and monitor actual QoS, and then store the profile information and historical behavior, e.g., successfully invoking a service, in the public cloud database. The observer agents also have to help gathering this information in the execution stage. According to the users' requirements, collaborative filtering is applied to find the appropriate web services, and to filter out the web services with incorrect QoS information. The web services which have correct QoS information and satisfy users' requirements are selected for the next phase. In the second phase, the QoS scores of the selected web services are derived from the correct QoS information and users' non-functional requirements. A high QoS score indicates that the web service meets the requirements of a user. Finally, the suitable web services with high QoS scores are recommended to the target users. Therefore, our proposed method not only satisfies users' QoS requirements without collecting any user ratings given to web services, but also addresses the shortcomings of the methods discussed in Section 1.

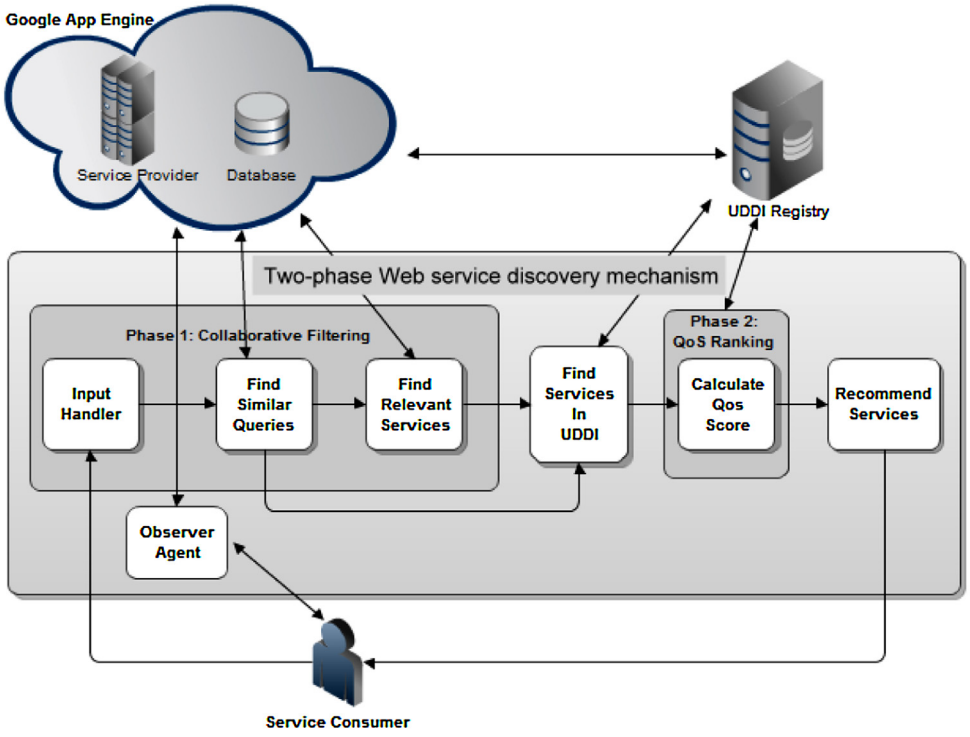

Fig. 3. The architecture of a trustworthy two-phase web service discovery mechanism.

3.1. Phase 1: collaborative filtering

The purpose of the first phase is to find the implicit relationships between historical queries, web services, and users; and to avoid the problems of malicious users and malicious attacks. In this phase, all user profiles will be analyzed, and then the web services with incorrect QoS information will be filtered out using the collaborative filtering algorithm. Moreover, the observer agents can help user to check the achievements of QoS which are claimed by service provider. Therefore, users do not need to report to the system. This helps users discover the web services that meet their functional requirements. There are three steps in the first phase: (1) establishing query and web services matrices, (2) calculating query similarity, and (3) calculating the relevance between a query and web services.

3.1.1. Step 1: establishing query and web services matrices

In this step, we establish a matrix of queries and web services (Fig. 4), where an entry represents the relationship between a query and a web service. Generally, a user queries and invokes the web services registered in UDDI. The matrix can be used to record the frequency that a web service is successfully invoked by a query. Let q_j be a query j that a user has requested, and w_i be a web service i which has been invoked by the user. M is the query-service matrix, where each row represents a query q_j, and each column represents a web service w_i. An entry I_{ij} in M is a frequency count indicating that the web service w_i has been invoked successfully by using the query q_j. That is, the relationship between q_j and w_i is described by using a frequency count I_{ij}. When such a relationship is built, the following two conditions must be checked and satisfied: First, a user must invoke a web service w_i successfully; Second, when the user invokes the web service w_i, its actual QoS value detected by the system must be equal to the QoS value provided by the service provider. If these two conditions are fulfilled, the frequency count I_{ij} is increased.

3.1.2. Step 2: calculating the query similarity

In this step, we calculate the similarity between a user's query and other queries in the matrix to find appropriate web services with highly similar queries. In this study, we apply the Vector Space Model (VSM) (Salton et al., 1975) to represent a query as a term vector. A query is represented as an n-dimensional vector comprised of the key terms and their respective weights, derived by the *tf-idf* approach. Based on the term weights, terms with higher values are selected as discriminative terms in order to describe the characteristics of the query. Let a query be $q_j =< t_{j1} : tw_{j1}, t_{j2} : tw_{j2}, \cdots t_{jx} : tw_{ix} >$, where $t_{jx} \in T, x = 1 \ldots s$, in which t_{jx} is a term x in query q_j, and belongs to a term set T derived from all queries Q where $Q = \{q_1, q_2, \ldots q_m\}$. The term weight tw_{jx} is the degree of importance of the term x to the query q_j, as defined in Eq. (1).

$$tw_{jx} = \frac{n_{jx}}{|n|} \times \log\left(\frac{m}{N}\right) \tag{1}$$

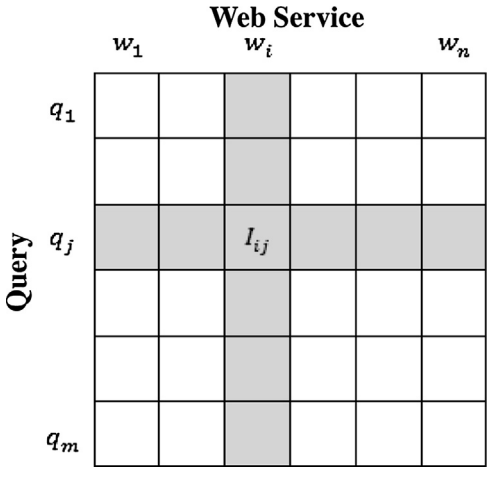

Fig. 4. Matrix of queries and web services.

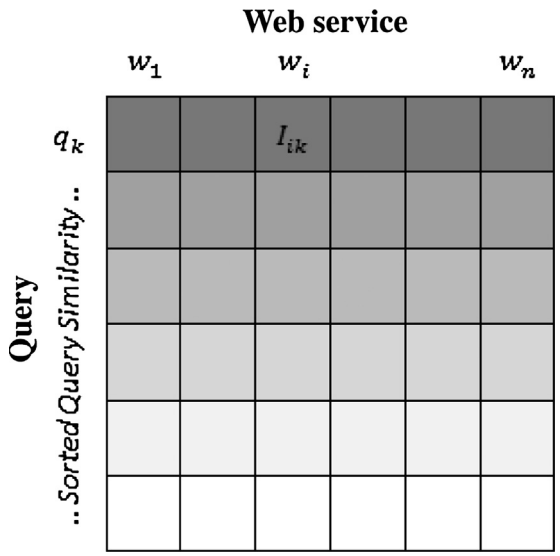

Fig. 5. Sorted matrix based on the query similarity.

where n_{jx} is the frequency of term x in query q_j, $|n|$ is the number of terms in query q_j, m is the total number of queries collected by the system, and N denotes the number of queries that contain the specific term n_{jx}. Therefore, we transform all the term x related to the query q_j to a weight vector $\vec{q}_j = (tw_{j1}, tw_{j2}, \ldots, tw_{jn})$. To identify similar query vectors, we evaluate the similarities between two query vectors using the cosine measurement, which computes the cosine of the angle between the queries' corresponding feature vectors. Two queries are regarded as similar if the cosine similarity value is high. The cosine similarity of two queries q_j and q_k is shown in Eq. (2).

$$sim(\vec{q}_j, \vec{q}_k) = \frac{\vec{q}_j \cdot \vec{q}_k}{|\vec{q}_j| \times |\vec{q}_k|} \tag{2}$$

Given a query q_k, the similarities between q_k and other queries q_j in a query-service matrix are measured by using Eq. (2). Then, based on these computation results, the entries in the query-service matrix, as described in step 1, are sorted from high to low, as shown in Fig. 5. That is, entries in the top rows of the query-service matrix are highly similar to the query q_k.

3.1.3. Step 3: calculating the relevance between query and web services

The third step in collaborative filtering is to calculate the relevance between a given query and web services, based on the sorted query-service matrix. Generally, a user employs a specific query to search some web services that may satisfy that user's individual or special demand. For example, if a user submits a query about "weather forecast", s/he could have a high possibility of choosing the web services related to "weather". Thus, to find highly relevant web services, the relevance measurement, as defined in Eq. (3), is used to calculate the relevance between a given query and the web services in the query-service matrix. The relevance formula is defined as follows:

$$\text{Relevance}(w_s, q_k) = \frac{I_{sk}}{\sum_{j=1}^{n} I_{jk}} \tag{3}$$

where q_k is a query in Q, w_s is one of the web services in a query-service matrix M, I_{sk} is a frequency count in matrix M representing how many times the web service w_s has been invoked successfully by using query q_k, and $\sum_{j=1}^{n} I_{jk}$ is the total frequency count for all services invoked by query q_k. The large value of I_{sk} means that the query q_k has invoked frequently the web service w_s successfully.

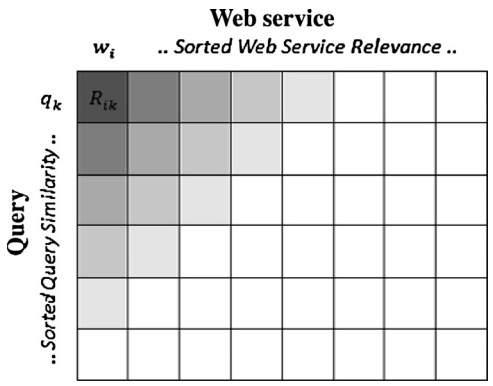

Fig. 6. The query-service matrix with sorted relevance degree.

That is, the web service has a high possibility of being chosen by a specific query. Thus, the web service and the specific query have a high relevance.

Next, we combine the relevance measurement with the query similarity in order to obtain the relevance degree. Let R_{ik} be the relevance degree between the web service w_i and a given query q_k, as shown in Eq. (4). Based on the relevance degrees, the values in the query-service matrix are sorted from high to low, as shown in Fig. 6.

$$R_{ik} = \text{Relevance}(w_i, q_j) \times sim(q_k, q_j), \tag{4}$$

The high relevance degree means that the web service is highly related to the query and the other relevant services are also discovered according to other similar queries. Conversely, if queries similar to the query q_k are difficult to find, there are few web services related to q_k. This leads to a very low possibility of discovering an appropriate web service for the user. Finally, we filter out the web services with relevance degrees lower than a pre-specified threshold. That is, the high relevant web services can be reserved and used in the next phase. As a result, the discovered web services have high probabilities of fulfilling users' functional requirement, and have correct QoS information from the first phase.

3.2. Phase 2: QoS ranking

This phase is mainly to identify web services that meet users' non-functional requirements from the discovered services in the first phase, and to resolve the problem of service selection. From the first phase, the discovered services may satisfy users' functional requirements, and have correct QoS information. However, these services may provide the same functions. Users may have difficulty choosing between them. Therefore, we propose the second phase to solve this problem. The second phase consists of 3 steps: (1) establishing a matrix of QoS and web services, (2) normalizing the QoS value, and (3) calculating the QoS score.

3.2.1. Step 1: establishing a QoS and web services matrix

In this study, we choose the following four common attributes for measuring web service QoS:

(1) *Price*: the cost of requesting a web service.
(2) *Availability*: the probability of finding an available web service within a period of time.
(3) *Response time*: the time taken for a user to receive a response after sending a request message to a web service.
(4) *Throughput:* how many requests a web service can handle within a period of time.

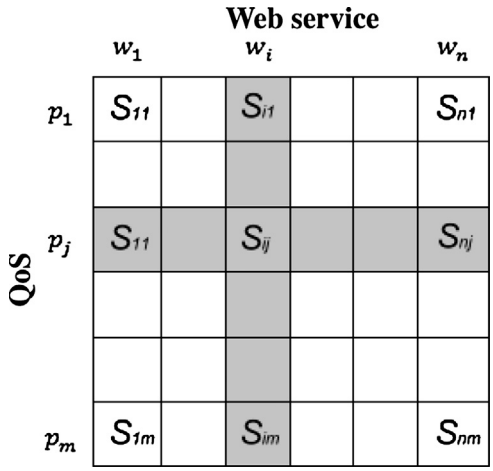

Fig. 7. The matrix of QoS attributes and web services.

First, we establish a matrix of QoS attributes and web services for all web services that provide the same function, as shown in Fig. 7. In the matrix, w_i represents a web service i, n is the number of web services, p_j is the attributes for measuring the QoS, m is the number of attributes, and S_{ij} is the value of QoS evaluated by the attribute p_j for web service w_i. In addition, because each QoS attribute has a different meaning, the QoS value S_{ij} stored in the matrix depends on the characteristic of QoS attribute p_j. Here, the QoS attributes are divided into two categories. One category is that the web services with small values of the QoS attributes have better quality. For example, a service has a very good quality because its price is low, or its response time is small. The other category is that the web services with large values of QoS attributes have better quality. For example, a service has a good quality because its availability is high, or its throughput is large. To deal with different values of attributes from these two categories, we transform the values of QoS attributes in the former category into the reciprocal of their values, and then put them into the matrix. For the attributes in the latter category, we directly keep their values in the matrix.

3.2.2. Step 2: normalizing the QoS value

Since the measuring unit and range of value for each QoS attribute are different, we need to normalize the values in the matrix of QoS and web services derived from step 1. Thus, all values can be transformed into the same scale. Then, they can be compared and ranked on the same basis. Based on the matrix in Fig. 7, we first have to find a maximum value of each QoS attribute p_j, and then normalize the value S_{ij}. The formula for normalization is shown in Eq. (5)

$$N_{ij} = \frac{S_{ij}}{\max\limits_{i=1\ldots n}(S_{ij})} \tag{5}$$

where N_{ij} is the normalized value for q_{ij}, $\max\limits_{i=1\ldots n}(S_{ij})$ is the maximum value of the QoS attribute p_j, and n is the number of web services. After the normalization, the values in the matrix are on a scale of 0–1.

3.2.3. Step 3: calculating the QoS score

To satisfy each user's QoS requirements, a QoS attribute p_j is given a different weighting WT_j when the QoS score is calculated. For example, when a user searches a low price web service, it indicates that the user prefers the "price" attribute to other QoS attributes. Thus, it is reasonable to give a large weight to this attribute.

The request message for querying a web service will contain the information about the attribute priorities in a user's requirements. The higher priority the requirement is, the higher the weight given to the attribute. For instance, if the attribute priorities in a user's requirements from high to low are price, response time, throughput, and availability, the highest weight is given to the "price" attribute. Notably, the value of each weight is in the range of 0–1, while the sum of all weightings is equal to 1. Thus, we define a formula in Eq. (6) for computing a QoS score QS_{ij} for web service w_i with an attribute p_j:

$$QS_{ij} = WT_j \times N_{ij} \tag{6}$$

where WT_j is a weight for the QoS attribute p_j, and N_{ij} is a normalized value of S_{ij}, described in step 2.

The web service has a large QoS score for a specific attribute if its attribute weight and normalized value are high. Finally, Eq. (7) is used to sum up the QoS scores of all QoS attributes for a specific web service. Let $QoSRank(w_i)$ be a ranking score of web service w_i. The scores of QoS attribute p_j are summarized as:

$$QoSRank(w_i) = \sum_{j=1}^{m} QS_{ij} \tag{7}$$

Such a score indicates that a web service with a high ranking score has a high probability of fulfilling a user's needs. Based on the computation result, web services with high QoS ranking scores can be compiled as a recommendation list, and then recommended to users. Therefore, our approach can recommend web services with good quality to users. If all the QoS attribute values are not available for all the services, this approach will still give a ranking list by the QoS score which resulted in the second phase. It can also reduce users' effort in searching web services, and help them to discover appropriate web services that satisfy their requirements.

4. Experiments and evaluations

In this work, we implement a system based on the proposed approach and platform, as shown in Fig. 3, and then develop experimental cases to evaluate and analyze the system's performance. In this experimental platform, jUDDI (version 2.0rc6) is a UDDI registry server, and the tool UDDI4J allows service providers to publish web service information and registries of QoS information to the tModel in a UDDI center. UDDI4J also allows service consumers to query the needed web services and QoS information.

According to the proposed method, the main functions of this system can be divided into three parts: (1) collaborative filtering module, (2) QoS ranking module, and (3) user behavior observation module. The first two modules respectively represent the first and second phases of the two-phase web service discovery mechanism. When using the proposed system platform, a service consumer can select a web service from a recommendation list, and then the system sends a request to a service provider. The third module, the user behavior observation module, will record and store not only requests and response messages between users and service providers, but also the actual QoS information into a cloud database on Google App Engine cloud platform. This approach implemented service providers by Google App Engine for simulation experiments.

For service users, their profile information are historical behavior (user queries), and historical invoking availability (successfully invoking a service). They will be recorded in the cloud database. When a new user sends a query to request a service, this system will provide a recommendation list generated from the database. All of these web services are provided in the Google App Engine cloud environment, so that everyone can use the web services, anywhere.

```
<tModel tModelKey="uuid:CD978180-3B13-11DF-A8EB-D2A3C1ECE15B">
    <name>QoS information for Weather service</name>
        <description xml:lang="en">
        Quality of Service Information for Weather service
        </description>
    <overviewDoc>
        <description xml:lang="en"></description>
            <overviewURL>
            http://<URL describing schema of QoS attributes>
            </overviewURL>
    </overviewDoc>
    <categoryBag>
        <keyedReference tModelKey=" uuid:CD978180-3B13-11DF-A8EB-D2A3C1ECE15B "
        keyName="Price" keyValue="0.01" />
        <keyedReference tModelKey=" uuid:CD978180-3B13-11DF-A8EB-D2A3C1ECE15B "
        keyName="Availability" keyValue="99.0" />
        <keyedReference tModelKey=" uuid:CD978180-3B13-11DF-A8EB-D2A3C1ECE15B "
        keyName="ResponseTime" keyValue="70" />
        <keyedReference tModelKey=" uuid:CD978180-3B13-11DF-A8EB-D2A3C1ECE15B "
        keyName="Throughput" keyValue="800" />
    </categoryBag>
</tModel>
```

Fig. 8. Example of publishing QoS information of weather web services.

4.1. Implementation for managing QoS information

In UDDI v3 architecture, *tModel* is used to describe the external technical specifications, or taxonomies, of web services, such as format, protocol, business logic, parameters about input and output. This provides a useful search method for service requesters, such that the service search is not limited to the business name, classification, or service name. The instance of *tModel* is presented as *keyReference*, in which *tModelKey* is an identified key, and *categoryBag* stores the classified information in UDDI. Thus, the *tModel* becomes a reference system to describe technical specifications.

In our system, the QoS information is stored by expanding the *tModel* in a UDDI registry center. Before a service provider publishes a web service, a corresponding *tModel* will be created to store web services' QoS information. Therefore, a service requester can query the web services' QoS information by using *tModelKey*. The QoS information consists of QoS attributes expressed by *KeyedReference* in *tModel*. For a QoS attribute, *keyName* represents the name of QoS attribute, while *keyValue* represents the QoS value. The QoS attributes' unit quantifiers are not expressed in *tModel*, so in this study we assume that these unit quantifiers have default values. This means that the unit measured by service providers and users is the same. For instance, Fig. 8 shows the QoS information of a weather web service described in a *tModel* is: (1) price: $0.01 per request, (2) availability: 99%, (3) response time: 70 ms, and (4) throughput: 800 requests per second. In summary, *tModel* will be created to describe the QoS information first, before a service provider publishes a web service to UDDI. It then uses *bindingTemplate* to reference the web service. The API (Application Programming Interface) of UDDI, e.g., UDDI4J, can be used to publish a web service.

When a service provider updates the QoS information, it can use the same *tModelKey* to modify and store the *tModel* content. Only the service provider has privileges to modify QoS information for its published services. Thus, the service provider should update QoS information for its owned web services regularly and often to ensure the correctness of information. The API of UDDI and UDDI4J can also help service providers update QoS information. The main steps are as follows:

Table 1
4 cases with different QoS orders.

Case	Quality of Service sequence
1	Price, Response time, Availability, Throughput
2	Response time, Price, Availability, Throughput
3	Availability, Response time, Price, Throughput
4	Throughput, Response time, Availability, Price

(1) Using *tModelKey* to find its corresponding *tModel* in UDDI.
(2) Modifying the value of *keyedReference* in *tModel*.
(3) Updating and storing QoS information to the same *tModelKey*.

4.2. Experiment design

In the experiments, we use simulations to validate the proposed service discovery method, because it is difficult to collect and summarize real web service data on the Internet for trustworthy QoS-based service discovery and selection problems. Some experimental cases and parameters are designed in the simulation to evaluate the performance of the proposed approach. The key in the experiments is to evaluate the search performance of the proposed trustworthy QoS-based collaborative filtering approach for service discovery and selection in all possible cases, with different QoS parameters. For evaluating the proposed approach, firstly, four experimental cases which take account of different QoS orders are designed in the experiment (i.e., Table 1), and then we give three different types of QoS status for QoS setting parameters (i.e., Table 2). The key in the experiment is to evaluate the search precision and ranking performance of the proposed trustworthy QoS-based collaborative filtering approach for service discovery.

Because it is difficult to collect and summarize real web service data on the Internet, we implement a system based on our proposed approach and platform, as shown in Fig. 3. We choose four common service categories (Weather, Stock, Currency, and Translation) and design four experimental cases with different orders of QoS attributes, as shown in Table 1. These four attributes, which are the common QoS information in the Service-Oriented Architecture (SOA) field, are price, response time, availability, and throughput. Response time and availability change dynamically, and are easily affected by other factors in a changing environment. Because the average response time for randomly initializing 200 services is 0.75 s, we use 0.75 as the value of "response time" as a "Better" status. In addition, each QoS attribute has different priority and importance for each user, so different weighting values are given for each QoS attribute. In our experiments, the weighting values of QoS attributes in a sequence are set to 0.6, 0.2, 0.15, and 0.05, respectively. For instance, the weighting values of QoS attributes in case 1 are 0.6 for price, 0.2 for response time, 0.15 for availability, and 0.05 for throughput.

There are three QoS statuses (Better, General, and Dishonest) in the experiments, to represent three different situations. "Better" means that the QoS of web service provided by a service provider has better performance and competitive advantage than that of other service providers. For instance, a web service for which QoS is a low price or a short response time has a "Better" status. "Dishonest" indicates that the actual QoS value of a web service provided by a service provider is different from the value that the service provider has claimed. In other words, the service provider claims that the response time must be within 2 ms, but in fact the response time is over 5 ms. The service's claimed value does not match its actual value. To evaluate the effectiveness of our approach, the QoS status and parameters of web services have to be designed in advance. The QoS attribute values, i.e., price, response time, availability, and throughput, are shown in Table 2. These QoS values are given in three status (better, general, and dishonest) for further design of a web services dataset in our simulation experiment. In the better status, we set the value of

Table 2
QoS setting values in 3 statuses.

QoS Status	Price	Promised response time	Real response time	Promised availability	Real availability	Throughput
Better	0.01	0.85	Around 0.75	0.99	1.0	800
General	0.03	1.3	Around 1.3	0.6	0.8	200
Dishonest	Not dishonest	Better 0.85 General 1.3	Around 2.0	Better 0.99 General 0.6	0.2	Not dishonest

promised response time to 0.85 s, but set the value of real response time in the system to 0.75 s. The response time in the general status is set to 1.3 s, and the real response time is around 1.3 s. For the dishonest status, the real response time is set to 2 s, which is always higher (dishonest) than the promised response time. Additionally, the promised availability value in a better status is set to 0.99, while the real availability is always successful (real availability = 1.0). For the general status, the promised availability value is set to 0.6, while the real availability value is set to 0.8. For the dishonest status, the real availability value would be 0.2, which is always lower (dishonest) than the promised availability.

Because we have to give all types of web services in the experimental design, Table 3 is the pre-defined guaranteed QoS status by service providers in four common service categories (Weather, Stock, Currency, and Translation). Each category has 24 web services, and every service has its specific guaranteed QoS information. In summary, 96 web services are used in the experiment.

We assumed the simulation platform in this section, and generated testing web services in Table 3. After the proposed trustworthy two-phase service discovery algorithm based on collaborative filtering and QoS, the system iteratively executed phase 1 (collaborative filtering) and phase 2 (QoS ranking) to filter out services with incorrect QoS values, while the second phase is employed to recommend services to users according to their non-functional requirements, which describe the correct QoS information for service consumers.

Define service claimed policies (if-then-else) for the status definitions:

(1) Claimed Price:
  If (Price = Better) Then (Price value = 0.01)
  If (Price = General) Then (Price value = 0.03)
(2) Claimed Response Time:
  If (Response Time = Better) Then (Response Time value = 0.85)
  If (Response Time = General) Then (Response Time value = 1.30)
(3) Claimed Availability:
  If (Availability = Better) Then (Availability value = 0.99)
  If (Availability = General) Then (Availability value = 0.6)
(4) Claimed Throughput:
  If (Throughput = Better) Then (Availability value = 800)
  If (Throughput = General) Then (Availability value = 200)

Define service real policies (if-then-else) for the status definitions:

In Honest Cases:

(1) Real Price:
  If (Price = Better) Then (Price value = 0.01)
  If (Price = General) Then (Price value = 0.03)
(2) Real Response Time:
  If (Response Time = Better) Then (Response Time value is around 0.75)
  If (Response Time = General) Then (Response Time value is around 1.30)
(3) Real Availability:
  If (Availability = Better) Then (Availability value = 1.0)
  If (Availability = General) Then (Availability value = 0.8)
(4) Real Throughput:
  If (Throughput = Better) Then (Availability value = 800)
  If (Throughput = General) Then (Availability value = 200)

In Dishonest Cases:

(1) Price and Throughput have no dishonest cases in the experiment design.
(2) Real Response Time:
  If (Response Time = Dishonest) Then (Real Response Time value is around 2.0)
(3) Real Availability:

If (Availability = Dishonest) Then (Availability value = 0.6)

4.3. Evaluation criteria

Because there was no standard benchmark for trustworthy QoS-based service discovery and selection problems, the evaluation criteria "ranking precision" is proposed as the indicator to evaluate and verify the recommendation results for web services. Such

Table 3
QoS benchmarks for 4 categories.

Categories: Weather, Stock, Currency, and Translation				
Web service	QoS information			
	Price	Response time	Availability	Throughput
WS_01–04	Better*	General	General	General
WS_05	Better*	General (*Dishonest*)	General	General
WS_06	Better*	General	General (*Dishonest*)	General
WS_07–10	General	Better*	General	General
WS_11	General	Better (Dishonest)*	General	General
WS_12	General	Better*	General (*Dishonest*)	General
WS_13–16	General	General	Better*	General
WS_17	General	General (*Dishonest*)	Better*	General
WS_18	General	General	Better (Dishonest)*	General
WS_19–22	General	General	General	Better*
WS_23	General	General (*Dishonest*)	General	Better*
WS_24	General	General	General (*Dishonest*)	Better*

* Significance.

Table 4
Example of a query result.

Service ranking for query *"current weather"*		
1. WS_3	9. WS_19	17. WS_8
2. WS_2	10. WS_6	18. WS_17
3. WS_7	11. WS_5	19. WS_14
4. WS_10	***12. WS_4***	20. WS_13
5. WS_18	***13. WS_1***	21. WS_24
6. WS_16	14. WS_12	22. WS_23
7. WS_15	15. WS_11	23. WS_21
8. WS_22	16. WS_9	24. WS_20

indicator not only meets users' functional requirements, but also has correct QoS information. In Eq. (8), let R be the expected rank of web services (as shown in Table 3), r be the correct rank of web services in a recommendation list, and N be the number of correct web services in a recommend list. Higher ranking precision means that the recommended services have high rank order, and correct ranks.

$$\text{Ranking Precision} = \frac{\sum R/r}{N} \tag{8}$$

Here is a simple example to describe the practicality of the approach. When a user wants to find a local weather request service with the cheapest price, the first phase is used to establish the past user queries and web services matrices, calculate query similarity, and calculate the relevance between query and web services. Then, the second phase is used to establish a matrix of QoS and web services, normalize the QoS value, and calculate the QoS scores. Finally, the system recommends services to users according to their non-functional requirements (cheapest price), which describe the correct QoS information for service consumers. According to the QoS benchmark in Table 3, the correct set of recommended web services is WS_1, WS_2, WS_3, and WS_4. Table 3 is used as a benchmark to compare with the web services in a recommendation list. If more recommended services meet the user's needs, and have high ranking values (i.e., small ranks), it means that our proposed method can recommend accurately. Thus, the ranking precision will be high. In this example, the user requests services for "current weather" and "cheapest", then our system recommends web services to the user, as shown in Table 4. From Table 4, the ranking priority of the correct web services in the recommendation list are 1, 2, 12, and 13, respectively (actual ranking priority is 1, 2, 3, 4). Actually, WS1, 2, 3, and 4 have the same rank, therefore, they can be put in the position 1, 2, 3, or 4 in the recommendation result (Ranking Precision = 1). In this brief example, WS4 is R = 3 and not R = 4 because it shows in the third position after WS3 and WS2. Based on the ranking values of services, the value of ranking precision for this query result is 0.64.

$$\text{Ranking Precision} = \frac{(1/1)+(2/2)+(3/12)+(4/13)}{4} = 0.64$$

4.4. Experiment results

For four experimental cases and four service categories, we send 300 service query requests separately, and then calculate their ranking precisions in order to evaluate the recommendation performance. The ranking precisions of four service categories in different experimental cases are shown in Fig. 9. The total ranking precisions for the four cases are 81%, 91%, 100%, and 81%, respectively. The comparison of each of the service categories with different cases is shown in Table 5. Based on Fig. 9, the average performance of these four experimental cases is presented in Fig. 10. With the increase in the number of requests, the ranking precision of all cases gradually rises to 88.25%. Furthermore, we compare the summary of

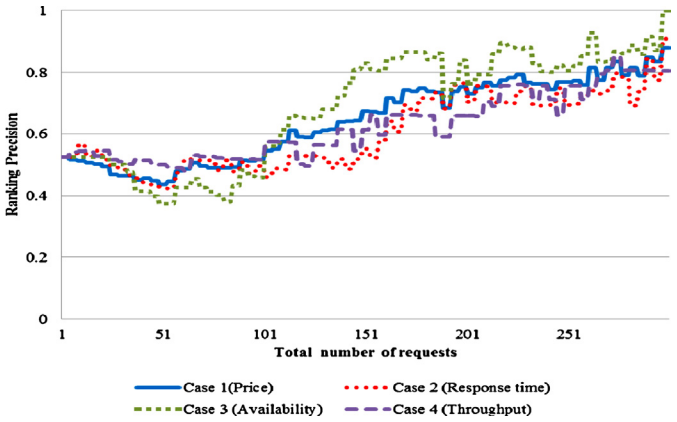

Fig. 9. Performance of 4 experimental cases.

Table 5
Ranking precision results of 4 experimental cases.

Categories/ experimental cases	Weather	Stock	Currency	Translation	Average
Case 1	82%	80%	81%	80%	81%
Case 2	82%	100%	100%	81%	91%
Case 3	100%	100%	100%	100%	100%
Case 4	82%	60%	81%	100%	81%
Average	87%	85%	91%	90%	88%

average performance of proposed approach (CF with QoS) with traditional collaboration filtering method without QoS consideration. In our experiment designs, we can obtain the real QoS information of service providers through the proposed approach.

In the experiments, with the more service requests, the more accurate the system recommends, because the collaborative filtering approach should show its effect. In contrast, if service providers give dishonest QoS information (it may be malicious or exceptional), the ranking precision results of the system recommendation for service discovery will decline in the learning period. However, in the proposed trustworthy two-phase service discovery algorithm based on collaborative filtering and QoS, it can further remove the influence of untrustworthy reasons. Moreover, we compare the proposed trustworthy QoS-based collaborative filtering approach with a traditional collaborative filtering approach by using the ranking precision. Consequently, the simulation experimental result shown in Fig. 10 indicates that the performance of this method can achieve an 88.25% accuracy rate, and its performance is significantly better than the traditional collaboration filtering method without QoS consideration.

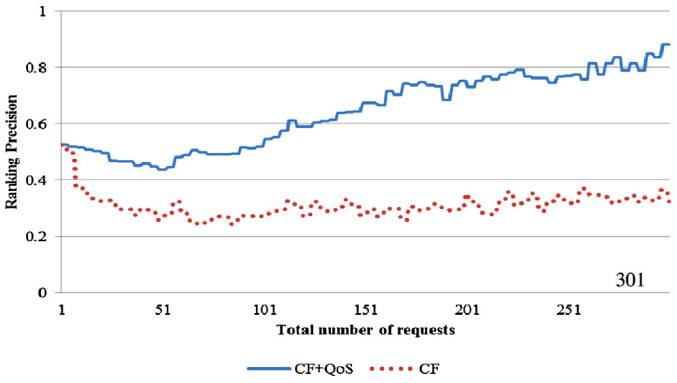

Fig. 10. Comparison of the average performance between the proposed approach and the traditional CF approach.

Therefore, web services with incorrect QoS information can be filtered out by the proposed method, and then web services are recommended according to users' QoS requirements. All in all, using the proposed method can help other users find the best web services more quickly through the users' profiles which included past query behavior. Since the specific design in these experimental cases, the more records of users' behavior, the more accurate recommended web services will be. The reason is the proposed approach can not only select and discover appropriate services via historical user profiles, but also filter out the services with dishonest QoS values assumed in Tables 2 and 3.

5. Conclusions

With the rapid development of web services technology and Cloud computing environments, more and more service providers supply web services with the same features. To solve the service discovery problem, we propose a trustworthy two-phase service discovery algorithm based on collaborative filtering and QoS in order to recommend good services from the same functional service group to users. In the recommendation process, our method can verify the correctness of QoS for web services. Therefore, the recommended services not only meet users' functional requirements, but also have correct QoS information. To evaluate our proposed method, we implement a system, and conduct experiments to compare our method with the traditional collaborative filtering method. From the experimental results, the performance of our method indicates that we can achieve an 88.25% average accuracy rate, and its performance is significantly better than the traditional collaborative filtering method. Obviously, the proposed approach can efficiently and effectively discover trustworthy web services, and accurately recommend the needed services to users. In future works, we will further employ users' usage data of their accessing web services to analyze users' behavior, and model user and service relationships in a social network. Then, a service recommendation method which considers both users' behavior and a social network will be proposed to provide service predictions for enhancing the recommendation performance. Moreover, we will combine this method with a semantic concept to build ontology of web services for describing the data relationship between web services in order to assist the data and knowledge discovery process for web services.

Acknowledgements

This work was supported in part by the National Science Council of the Taiwan [grant numbers NSC 99-2410-H-009-036-MY3 and NSC 100-2410-H-033-037].

References

Al-Masri, E., Mahmoud, Q.H., 2007. QoS-based discovery and ranking of web services. In: Proceedings of the 16th International Conference on Computer Communications and Networks (ICCCN'07), Honolulu, Hawaii, pp. 529–534.

Ali, A.S., Majithia, S., Rana, O.F., Walker, D.W., 2006. Reputation-based semantic service discovery. Concurrency and Computation: Practice & Experience 18, 817–826.

Birukou, A., Blanzieri, E., D'Andrea, V., Giorgini, P., Kokash, N., 2007. Improving web service discovery with usage data. IEEE Software 24, 47–54.

Canfora, G., Di Penta, M., Esposito, R., Villani, M.L., 2008. A framework for QoS-aware binding and re-binding of composite web services. Journal of Systems and Software 81, 1754–1769.

Comuzzi, M., Pernici, B., 2009. A framework for QoS-based Web service contracting. ACM Transactions on the Web 3, 1–52.

Fernandez, A., Hayes, C., Loutas, N., Peristeras, V., Polleres, A., Tarabanis, K., 2008. Closing the service discovery gap by collaborative tagging and clustering techniques. In: Proceedings of 2nd Internatioal Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web, Karlsruhe, Germany, pp. 115–128.

González-Valenzuela, S., Vuong, S.T., Leung, V.C.M., 2011. Leveraging service discovery in MANETs with mobile directories. Computer Journal 55, 218–231.

Gouscos, D., Kalikakis, M., Georgiadis, P., 2003. An approach to modeling Web service QoS and provision price. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops, Roma, Italy, pp. 121–130.

Haas, H., Brown, A., 2004. Web Services Glossary. W3C Working Group http://www.w3.org/TR/ws-gloss/

Harshavardhanan, P., Akilandeswari, J., Sarathkumar, R., 2012. Dynamic Web Services discovery and selection using QoS-Broker architecture. In: International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, pp. 1–5.

Herlocker, J.L., Konstan, J.A., Riedl, J., 2000. Explaining collaborative filtering recommendations. In: Proceedings of the ACM Conference on Computer Supported Cooperative Work, Philadelphia, PA, United States. ACM, pp. 241–250.

Huang, A.F.M., Lan, C.-W., Yang, S.J.H., 2009. An optimal QoS-based Web service selection scheme. Information Sciences 179, 3309–3322.

Keller, A., Ludwig, H., 2003. The WSLA framework: specifying and monitoring service level agreements for web services. Journal of Network and Systems Management 11, 57–81.

Kerrigan, M.,2006. Web service selection mechanisms in the Web Service Execution Environment (WSMX). In: Proceedings of the 2006 ACM symposium on Applied computing. ACM, Dijon, France, pp. 1664–1668.

Khapre, S., Chandramohan, D., 2011. Personalized web service selection. International Journal of Web & Semantic Technology 2, 78–93.

Ko, J.M., Kim, C.O., Kwon, I.-H., 2008. Quality-of-service oriented web service composition algorithm and planning architecture. Journal of Systems and Software 81, 2079–2090.

Koloniari, G., Pitoura, E., 2004. Filters for XML-based service discovery in pervasive computing. Computer Journal 47, 461–474.

Konstan, J.A., Miller, B.N., Maltz, D., Herlocker, J.L., Gordon, L.R., Riedl, J., 1997. GroupLens: applying collaborative filtering to usenet news. Communications of the ACM 40, 77–87.

Li, L.Y., Liang, P.S., Xi, H., 2012. A kind of web service recommendation method based on improved hybrid collaborative filtering. In: IEEE 11th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC), pp. 103–108.

Liangzhao, Z., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H., 2004. QoS-aware middleware for Web services composition. IEEE Transactions on Software Engineering 30, 311–327.

Manikrao, U.S., Prabhakar, T.V.,2005. Dynamic selection of web services with recommendation system. In: Proceedings of the International Conference on Next Generation Web Services Practices (NWeSP). IEEE Computer Society, Seoul, Korea, pp. 117–121.

Martin-Diaz, O., Ruiz-Cortes, A., Corchuelo, R., Toro, M., 2003. A framework for classifying and comparing web services procurement platforms. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW), Roma, Italy, pp. 156–164.

Maximilien, E.M., Singh, M.P., 2004. A framework and ontology for dynamic Web services selection. IEEE Internet Computing 8, 84–93.

Maximilien, E.M., Singh, M.P., 2005. Self-adjusting trust and selection for web services. In: Proceedings of the Second International Conference on Autonomic Computing (ICAC'05), Seattle, Washington, pp. 385–386.

Menasce, D.A., 2002. QoS issues in Web services. IEEE Internet Computing 6, 72–75.

Mokhtar, S.B., Preuveneers, D., Georgantas, N., Issarny, V., Berbers, Y., 2008. EASY: efficient semantic service discovery in pervasive computing environments with QoS and context support. Journal of Systems and Software 81, 785–808.

Mukhopadhyay, D., Chougule, A., 2012. A survey on web service discovery approaches. In: Wyld, D.C., Zizka, J., Nagamalai, D. (Eds.), Advances in Computer Science, Engineering & Applications. Springer, Berlin, Heidelberg, pp. 1001–1012.

Nair, M.K., Gopalakrishna, V., 2010. Look before you leap: a survey of web service discovery. International Journal of Computer Applications 7, 22–30.

Papazoglou, M.P., 2003. Service-oriented computing: concepts, characteristics and directions. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE'03), Roma, Italy, pp. 3–12.

Raj, R.J.R., Sasipraba, T., 2010. Web service selection based on QoS Constraints. In: Trendz in Information Sciences & Computing (TISC), Chennai, India, pp. 156–162.

Ran, S., 2003. A model for web services discovery with QoS. ACM SIGecom Exchanges 4, 1–10.

Salton, G., Wong, A., Yang, C.S., 1975. A vector space model for automatic indexing. Communications of ACM 18, 613–620.

Sarwar, B., Karypis, G., Konstan, J., Reidl, J.,2001. Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th International Conference on World Wide Web. ACM, Hong Kong, pp. 285–295.

ShaikhAli, A., Rana, O.F., Al-Ali, R., Walker, D.W.,2003. UDDIe: an extended registry for web services. In: Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops). IEEE Computer Society, Orlando, Florida, pp. 85–89.

Torres, R., Astudillo, H., Salas, R., 2011. Self-adaptive fuzzy QoS-driven web service discovery. In: IEEE International Conference on Services Computing (SCC), pp. 64–71.

Wang, J., Vries, A.P.d., Reinders, M.J.T.,2006. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, Seattle, Washington, USA, pp. 501–508.

Wang, J., Vries, A.P.d., Reinders, M.J.T., 2008. Unified relevance models for rating prediction in collaborative filtering. ACM Transactions on Information Systems (TOIS) 26, 1–42.

Wishart, R., Robinson, R., Indulska, J., Jøsang, A.,2005. SuperstringRep: reputation-enhanced service discovery. In: Proceedings of the Twenty-Eighth Australasian conference on Computer Science. Australian Computer Society, Inc., Newcastle, Australia, pp. 49–57.

Xu, Z.Q., Martin, P., Powley, W., Zulkernine, F., 2007. Reputation-enhanced QoS-based web services discovery. In: Proceedings of the 2007 IEEE International Conference on Web Services (ICWS'07), Salt Lake City, Utah, USA, pp. 249–256.

Yan, J., Piao, J.T., 2009. Towards QoS-based web services discovery. In: Service-Oriented Computing – ICSOC 2008 Workshops, Sydney, Australia, pp. 200–210.

Yang, Y., Dumas, M., García-Bañuelos, L., Polyvyanyy, A., Zhang, L., 2012. Generalized aggregate Quality of Service computation for composite services. Journal of Systems and Software 85, 1818–1830.

Zheng, Z., Ma, H., Lyu, M.R., King, I., 2009. WSRec: a collaborative filtering based web service recommender system. In: Proceedings of the IEEE International Conference on Web Services (ICWS), Los Angeles, CA, USA, pp. 437–444.

Szu-Yin Lin is an assistant professor in the Department of Information Management at Chung Yuan Christian University, Taiwan. He received his Ph.D. degree in Information Management from National Chiao Tung University, Taiwan, in 2012. His expertise is on service-oriented architecture and computing, intelligent dynamic data analysis, multi-agent systems, and Internet technology.

Chin-Hui Lai is an assistant professor of the Department of Information Management, Chung Yuan Christian University, Taiwan. She received her B.S. in Department of Information Management from the National Taiwan University of Science and Technology in 2001. Then, she received her M.S. and Ph.D. degrees in Information Management from National Chiao Tung University, Taiwan, in 2004 and 2009. Her research interests include data mining, recommender systems, knowledge management and electronic commerce.

Chih-Heng Wu received his M.S. degree in the Institute of Information Management from National Chiao Tung University in Taiwan. Her expertise is on service-oriented computing and network communication.

Chi-Chun Lo is a professor in the Institute of Information Management at National Chiao Tung University in Taiwan. He received his Ph.D. degree in Computer Science from Brooklyn Polytechnic University, USA in 1987. His research interests include network management, network security, network architecture, and wireless communications.