Time Aware and Data Sparsity Tolerant Web Service Recommendation Based on Improved Collaborative Filtering

Yan Hu, Qimin Peng, Xiaohui Hu, and Rong Yang

Abstract—With the incessant growth of web services on the Internet, how to design effective web service recommendation technologies based on Quality of Service (QoS) is becoming more and more important. Web service recommendation can relieve users from tough work on service selection and improve the efficiency of developing service-oriented applications. Neighborhood-based collaborative filtering has been widely used for web service recommendation, in which similarity measurement and QoS prediction are two key issues. However, traditional similarity models and QoS prediction methods rarely consider the influence of time information, which is an important factor affecting the QoS performance of web services. Furthermore, it is difficult for the existing similarity models to capture the actual relationships between users or services due to data sparsity. The two shortcomings seriously devalue the performance of neighborhood-based collaborative filtering. In this paper, the authors propose an improved time-aware collaborative filtering approach for high-quality web service recommendation. Our approach integrates time information into both similarity measurement and QoS prediction. Additionally, in order to alleviate the data sparsity problem, a hybrid personalized random walk algorithm is designed to infer indirect user similarities and service similarities. Finally, a series of experiments are provided to validate the effectiveness of our approach.

Index Terms—Web service recommendation, collaborative filtering, time information, data sparsity, hybrid personalized random walk

1 Introduction

In recent years, Service-Oriented Computing (SOC) [1] has attracted a lot of attention as a new promising computing paradigm for software engineering and distributed computing [2]. In this paradigm, web services are basic constructs. They encapsulate application functionalities and make themselves accessible through standard interfaces, thus allowing interoperable machine-to-machine interaction over a network. According to recent statics [3], there are 28,606 web services available on the Internet as of 07/01/2013, provided by 7,739 providers.

In service-oriented systems, service users can construct web applications by organising a series of abstract tasks in a particular order to meet their personalized needs. Before execution, each abstract task should be bound to a concrete web service which meets the functional requirements of this task. And with the proliferation of functionally equivalent web services, we need additional information to differentiate them. Quality of Service (QoS) describes the non-functional characteristics of web services, including response time, throughput, availability, etc. It has been employed as an important factor to differentiate web services which provide similar functionalities [4]. However, it is not an easy task for service users to identify an optimal web service within a large set of functionally equivalent service candidates, especially when a web application involves many abstract tasks. QoS-based web service recommendation can solve this problem elegantly, by selecting a web service with optimal QoS performance for each abstract task. It can greatly improve the efficiency of developing web applications, so web service recommendation is a key issue in the field of service computing [1]. However, a major challenge faced by service recommendation is that service providers rarely deliver the QoS as declared, due to:

- Non-functional performance of web services is highly correlated with invocation time, since the service status (e.g., number of clients, workload, etc.) and the network condition (e.g., bandwidth, etc.) change over time.
- Service users are typically geographically distributed. The QoS performance of web services observed by users is highly influenced by the network connections between users and web services. Different users may observe totally different QoS performance even if they invoke a same web service.

Based on the above analysis, making time-aware personalized QoS prediction is important for high-quality web service recommendation.

In recent literature, neighborhood-based Collaborative Filtering (CF), which mainly includes user-based [5], [6], item-based [7], [8] and hybrid [9], [10] approaches, has

- *Y. Hu is with the University of Chinese Academy of Sciences, and the Science and Technology on Integrated Information System Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China. E-mail: huyan@iscas.ac.cn.*
- *Q. Peng and X. Hu are with the Science and Technology on Integrated Information System Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China. E-mail: {qimin, hxh}@iscas.ac.cn.*
- *R. Yang is with the College of Computer Science and Technology, Hubei University of Science and Technology, Xianning, China and the State Key Laboratory of Software Engineering and School of Computer, Wuhan University, Wuhan, China. E-mail: yr80427@whu.edu.cn.*

Manuscript received 16 Oct. 2014; accepted 2 Dec. 2014. Date of publication 17 Dec. 2014; date of current version 9 Oct. 2015.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSC.2014.2381611

become the most popular technology for personalized QoS prediction. These approaches make QoS prediction by collecting information from similar users or services. Furthermore, to improve the basic CF methods, some researchers propose to take context information into consideration for more accurate web service recommendation. The most discussed context information is location [11], [12], [13]. However, to the best of our knowledge, none of existing context-aware web service recommendation approaches relying on neighborhood-based CF consider the temporal dynamics of QoS, which is also an important context factor. This results in biased QoS prediction and greatly limits the performance of web service recommendation.

Apart from the lack of consideration of time information, another drawback of neighborhood-based CF is that it cannot handle data sparsity well. The number of users and services on the Internet is very large. Even very active users may invoke only a few web services and even very popular web services may be invoked by only a few users. This problem, commonly referred to as the *data sparsity problem* [14], has greatly limited the applicability of neighborhood-based CF, since it is difficult to discover enough similar users and services for QoS prediction based on sparse training data. Recently, the random walk mechanism has been widely discussed to alleviate data sparsity [15], [16], [17], [18]. Its basic idea is to construct a graph by treating items as nodes and the relationships between items as edges, and then assign a stationary visiting probability to each candidate item as its ranking score. Items with higher ranking scores are more likely to be recommended to users. However, web service recommendation is a multi-criteria decision-making procedure [19]. We should predict the missing value of each QoS property for a candidate web service. Thus the follow-up decision-making procedure can merge the values of different QoS properties into a comprehensive ranking score, rather than simply assign a single-value ranking score to each web service through random walk. Motivated by the existing random walk algorithms and the features of web services, we propose a hybrid personalized random walk algorithm to handle data sparsity for web service recommendation. It performs personalized random walk on both the user graph and the service graph to identify more similar neighbors for the target user, who requests web service recommendation currently and each candidate web service, respectively. Thus, we can extract enough historical information from these similar neighbors to make more accurate QoS prediction.

The key contributions of our work are three-fold:

- Time information is integrated into the similarity measurement and the QoS prediction of traditional neighborhood-based CF, for higher-quality web service recommendation.
- A hybrid personalized random walk algorithm is employed to handle data sparsity. It performs personalized random walk on both the user graph and the service graph, to discover more indirect similar users and services.
- A series of experiments are conducted over a freely available dataset, to validate the effectiveness of our time-aware approach combined with the hybrid personalized random walk mechanism for web service recommendation.

The rest of this paper is organized as follows. Section 2 presents some related work. Section 3 first states the problem and then presents an overview of the improved web service recommendation approach. In Section 4, we propose a time-aware similarity model. In Section 5, a hybrid personalized random walk algorithm is employed to handle data sparsity. Section 6 discusses how to make time-aware QoS prediction. Section 7 discusses the computational complexity of our algorithm. Section 8 shows the experiments. Finally, Section 9 concludes the paper.

2 Related Work

This section briefly presents an overview of research literature about web service recommendation, CF algorithms and data sparsity problems.

The increasing number of functionally equivalent web services on the Internet calls for effective QoS-aware web service recommendation technologies. Based on QoS performance of web services, various approaches [9], [10], [20], [21] have been proposed for web service recommendation. They enable the optimal web service to be identified from a set of functionally similar or equivalent web service candidates. Since the QoS performance of web services is generally not predetermined, CF algorithms have been widely utilized to make QoS prediction. Two types of CF approaches are widely studied: neighborhood-based and model-based. The most analyzed methods of neighborhood-based CF include user-based [5], [6], item-based [7], [8] and hybrid [9], [10] approaches. The user-based method utilizes historical QoS experience from a group of similar users to make QoS prediction, while the item-based method uses historical QoS information from similar services for QoS prediction. The hybrid approach is the combination of the user-based and service-based methods, which can achieve a higher QoS prediction accuracy. Additionally, the three methods often use Pearson Correlation Coefficient (PCC) as their similarity models. And almost all other similarity computation methods (e.g., cosine measure, adjusted cosine measure, constrained PCC, etc.) of neighborhood-based CF are discussed in [22]. Model-based CF mainly utilizes training data to train a predefined model and then uses the trained model for QoS prediction. The most discussed models mainly includes clustering models [23], aspect models [24] and latent factor models [25]. Our web service recommendation approach focuses on the neighborhood-based CF methods since they are more intuitive to interpret service recommendation results.

Furthermore, to achieve better prediction performance, other researchers incorporate context information into the basic CF methods. The recently most discussed context information is location [11], [12], [13]. They hold the opinion that the geographically close users or services usually have similar QoS experience. Thus, more historical QoS information from geographically close neighbors can be used for more accurate QoS prediction. However, to the best of our knowledge, none of the existing web service recommendation methods based on neighborhood-based CF consider the service invocation time information. Time is another

	s_1	s_2	$\cdots$	s_n
u_1	(q_{11}, t_{11})	(q_{12}, t_{12})	$\cdots$	(q_{1n}, t_{1n})
u_2	(q_{21}, t_{21})	(q_{22}, t_{22})	$\cdots$	(q_{2n}, t_{2n})
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
u_m	(q_{m1}, t_{m1})	(q_{m2}, t_{m2})	$\cdots$	(q_{mn}, t_{mn})

Fig. 1. Time-aware user-service matrix Q.

important context factor affecting QoS performance, since QoS performance of web services is highly related to some time-varying factors (e.g., service status, network condition, etc.). Therefore, no consideration of time information undoubtedly limits the QoS prediction accuracy.

Another drawback of neighborhood-based CF is that it is vulnerable to data sparsity. The indirect similarity inference approach [14] focuses on inferring indirect relationship between any two given users through the intermediate users located on the paths between the two users to alleviate data sparsity. However, this method cannot automatically recognize indirect similar neighbors for a given user. Other researchers employ random walk algorithms for automatic discovery of related objects. Yildirim and Krishnamoorthy [15] perform random walk on an item graph (constructed by treating items as nodes and item similarities as edges) to find more related items, and assign a stationary visiting probability to each item as its ranking score. This approach can alleviate data sparsity to some extent, but does not take user information into consideration. In [16], [26], [17], [18], random walk is performed on a hybrid graph, which is constructed by treating both users and items as nodes and the user-item interactions as edges. Similarly, items with higher stationary visiting probabilities are more likely to be recommended to target users. These methods based on random walk are applicable to one-dimensional rating systems. However, a web service recommender system is a multi-criteria decision-making system, in which users should evaluate the performance of a web service based on more than one QoS property, rather than simply assign a single-value ranking score to each candidate service during the random walk.

In this paper, we build a user graph and a service graph, whose edges are weighted by user similarities and service similarities, respectively. The random walk algorithm is performed on both of them to discover more indirect similar neighbors for the target user and each candidate web service. Thus, plentiful historical information extracted from these similar neighbors can yield a higher QoS prediction accuracy. Finally, the user-based and service-based prediction results are effectively merged into a comprehensive prediction result. Our method can apply to different QoS properties, thus to support the follow-up multi-criteria decision-making procedure for web service recommendation.

3 Problem and Approach Overview

This section includes the problem definition and a concise introduction to the entire procedure of the innovative web service recommendation approach, which is enhanced by the time-aware and data sparsity tolerant mechanism.

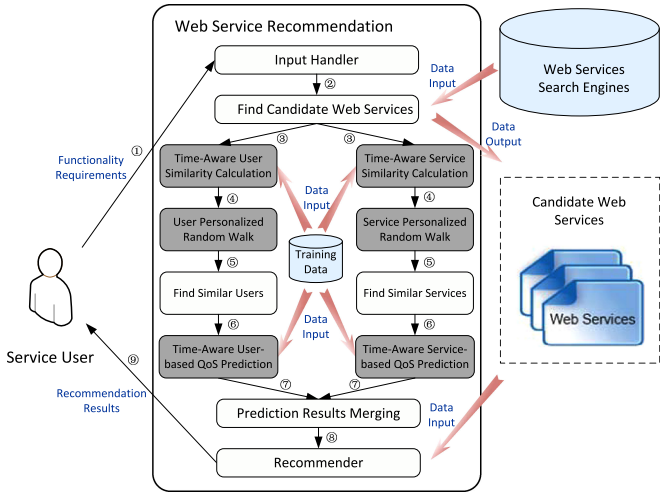


Fig. 2. Procedures of web service recommendation.

3.1 Problem Statement

Suppose that a web service recommender system contains a set of users $U = \{u_1, u_2, \ldots, u_m\}$ and a set of web services $S = \{s_1, s_2, \ldots, s_n\}$. The user-service relationships are denoted by a $m \times n$ matrix called the user-service matrix Q, as shown in Fig. 1. Each entry of this matrix is such a couple $(q_{ij}, t_{ij})(1 \leq i \leq m, 1 \leq j \leq n)$, where q_{ij} is the QoS value observed by user u_i on service s_j, and t_{ij} is the timestamp when u_i invoked s_j. For simplicity, we assume that a user invoked a service at most once. If u_i never invoked s_j, we have $q_{ij} = null$ (a missing QoS value). Otherwise, the observed QoS value and its corresponding timestamp are recorded in Q.

If a target user u_i requests web service recommendation by providing some functional requirements, the system should recommend an optimal web service to him. The recommended web service should not only provide the required functionalities but also have optimal QoS performance. Consequently, the missing QoS value q_{ij} should be predicted for each pair (u_i, s_j) (s_j is a candidate web service) at the current time $t_{current}$ (the time when the recommendation is requested), based on the historical QoS information contained in Q.

3.2 Overview of Our Approach

In order to address the problem stated above, we propose an improved web service recommendation approach. All procedures involved in this web service recommendation approach are depicted in Fig. 2. First, a target user requests web service recommendation by providing his functional requirements through the *Input Handler*. Then the system gets a set of candidate web services which meet the required functionalities through some *web services search engines*, such as *web service list*[1] and *Programmableweb*.[2] After that, time-aware similarities are computed for users and services based on historical training data. Afterwards, the personalized random walk mechanism is employed to infer indirect user similarities and service similarities. Next, similar neighbors are identified for the target user and each candidate service.

1. http://www.webservicelist.com
2. http://www.programmableweb.com

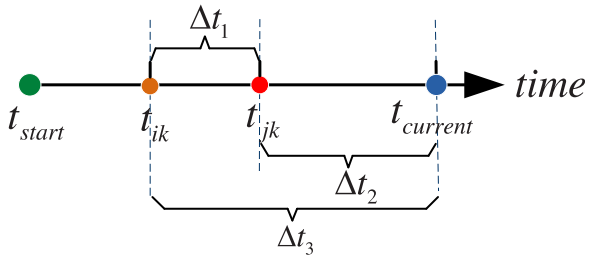


Fig. 3. Time factors affecting similarity measurement.

Following this, time-aware user-based and service-based QoS predictions are made. The user-based and service-based prediction results are merged together. Then the *Recommender* evaluates each candidate web service according to its predicted values of different QoS properties by employing some multi-criteria decision-making methods. Finally, the recommendation result is generated and returned to the target user.

Our contributions are highlighted in dark gray in Fig. 2. In the next three sections of this paper, we elaborate on how to calculate time-aware similarities, how to utilize personalized random walk mechanism to alleviate data sparsity and how to make time-aware QoS prediction.

4 Time-Aware Similarity Measurement

Similarity measurement is a key issue of neighborhood-based CF, since more accurate similarity measurement generally yields more accurate QoS prediction [9]. However, to the best of our knowledge, all existing similarity models mentioned in Section 2 neglect the influence of service invocation time. Time is an influential context factor because the QoS performance of web services is highly related to service invocation time due to the time-varying service status and network environment. Generally speaking, a longer timespan indicates a higher probability that a QoS value deviates from its original value.

We now elaborate on the time-aware similarity measurement for users, and that for services is analogous. First, we assume that a user can finish a service invocation in a very short timespan, which can be regarded as an instant. This assumption is reasonable, since compared with the entire life cycle of a web service recommender system, the duration of a service invocation is short enough to be ignored. There are two intuitive principles behind the user similarity measurement:

1) *Temporally closer QoS experiences from two users on a same web service contribute more to the user similarity value.*

As shown in Fig. 3, we assume that the web service recommender system starts at t_{start}. A user u_i invoked a web service s_k at t_{ik}, and another user u_j invoked the same service s_k at t_{jk}. The timespan between t_{ik} and t_{jk} is denoted by $\triangle t_1$. If $\triangle t_1$ is long, even though u_i and u_j have very similar QoS experiences on s_k, it does not really mean high similarity between u_i and u_j, since u_i's QoS experience on s_k may change violently over $\triangle t_1$. Therefore, a shorter $\triangle t_1$ (temporally closer) generally indicates a greater contribution of s_k to the similarity computation between u_i and u_j. Thus, the contribution of s_k can be approximately weighted by a exponential decay function of t_{ik} and t_{jk}, which is defined as

$$f_1(t_{ik}, t_{jk}) = e^{-\alpha|t_{ik}-t_{jk}|}, \tag{1}$$

where $\alpha \geq 0$ is a non-negative decay constant, with a larger α making the value of f_1 (ranging from 1 to 0) vanish more rapidly with the increase of the timespan $|t_{ik} - t_{jk}|$.

2) *More recent QoS experiences from two users on a same web service contribute more to the user similarity value.*

The similarity between two users at the current time generally depends more on their recent QoS experiences. Namely, if two users invoked a same web service a long time ago, their QoS experiences on this co-invoked service should be less considered while computing their similarity at the current time. As illustrated in Fig. 3, $\triangle t_2$ is the timespan between u_j's invocation of s_k and the current moment, and $\triangle t_3$ is the timespan between u_i's invocation of s_k and the current moment. We utilize $\triangle t_4 = (\triangle t_2 + \triangle t_3)/2$ (the average of $\triangle t_2$ and $\triangle t_3$) to denote the second time factor. A shorter $\triangle t_4$ (more recent) generally indicates a greater contribution of s_k to the user similarity value. Thus, we consider that the contribution of s_k also decays exponentially with the increase of $\triangle t_4$. This exponential decay function is defined as

$$f_2(t_{ik}, t_{jk}) = e^{-\beta|t_{current}-(t_{ik}+t_{jk})/2|}, \tag{2}$$

where $\beta \geq 0$ is a non-negative decay constant, with a larger β making f_2 (ranging from 1 to 0) vanish much more rapidly with the increase of the timespan $|t_{current} - (t_{ik} + t_{jk})/2|$.

Here, we take PCC as an example to describe the time-aware similarity model. Time information can be integrated into other similarity models listed in [22] in a similar way. Based on the definitions of f_1 and f_2, the time-aware PCC-based similarity measurement between two users u_i and u_j can be defined as

$$w_{ij}^u = \frac{\sum_{s_k \in S}(q_{ik} - \bar{q}_i)(q_{jk} - \bar{q}_j) f_1(t_{ik}, t_{jk}) f_2(t_{ik}, t_{jk})}{\sqrt{\sum_{s_k \in S}(q_{ik} - \bar{q}_i)^2}\sqrt{\sum_{s_k \in S}(q_{jk} - \bar{q}_j)^2}}, \tag{3}$$

where $S = S_{u_i} \cap S_{u_j}$ is the subset of web services which have been invoked by both u_i and u_j , and $\bar{q}_i$ and $\bar{q}_j$ are the average QoS values observed by u_i and u_j, respectively.

In reality, there may exist some complicate situations: a service user invoked a web service more than once. Although, under the background of data sparsity, a user rarely invoked a web service more than once, we still need to take this special case into consideration in order to promote the universality of our method. In this case, the corresponding entry in the time-aware user-service matrix $\boldsymbol{Q}$ is a set of couples instead of just one couple. For example, if u_i invoked s_k total P_{ik} times and u_j invoked s_k total P_{jk} times, the (i,k)th and (j,k)th entries of $\boldsymbol{Q}$ are two sets denoted by $Q_{ik} = \{(q_{ik}^{p_i}, t_{ik}^{p_i}) | 1 \leq p_i \leq P_{ik}\}$ and $Q_{jk} = \{(q_{jk}^{p_j}, t_{jk}^{p_j}) | 1 \leq p_j \leq P_{jk}\}$, respectively. Here, $q_{ik}^{p_i}$ and $t_{ik}^{p_i}$ are the QoS value and timestamp of u_i's p_ith invocation of s_k, and $q_{jk}^{p_j}$ and $t_{jk}^{p_j}$ are the QoS value and timestamp of u_j's p_jth invocation of s_k. Accordingly, the similarity measurement between u_i and u_j should be modified. While computing their similarity, each pair of $((q_{ik}^{p_i}, t_{ik}^{p_i}), (q_{jk}^{p_j}, t_{jk}^{p_j}))$ should be considered. Therefore, for each co-invoked web service s_k, we should consider total $P_{ik}P_{jk}$ pairs of $((q_{ik}^{p_i}, t_{ik}^{p_i}), (q_{ik}^{p_j}, t_{ik}^{p_j}))$ and

denote them by $\{((q_{ik}^{(p)}, t_{ik}^{(p)}), (q_{jk}^{(p)}, t_{jk}^{(p)}))|(q_{ik}^{(p)}, t_{ik}^{(p)}) \in Q_{ik}, (q_{jk}^{(p)}, t_{jk}^{(p)}) \in Q_{jk}, 1 \le p \le P_{ik}P_{jk}\}$. The modified time-aware user similarity measurement is shown in Eq. (4)

$$w_{ij}^{u} = \sum_{s_k \in S} \left(\frac{1}{P_{ik}P_{jk}} \sum_{p=1}^{P_{ik}P_{jk}} \left(q_{ik}^{(p)} - \bar{q}_i\right)\left(q_{jk}^{(p)} - \bar{q}_j\right) \times f_1\left(t_{ik}^{(p)}, t_{jk}^{(p)}\right) f_2\left(t_{ik}^{(p)}, t_{jk}^{(p)}\right) \right) \Bigg/ \left(\sqrt{\sum_{s_k \in S} \frac{1}{P_{ik}} \sum_{p_i=1}^{P_{ik}} \left(q_{ik}^{p_i} - \bar{q}_i\right)^2} \times \sqrt{\sum_{s_k \in S} \frac{1}{P_{jk}} \sum_{p_j=1}^{P_{jk}} \left(q_{jk}^{p_j} - \bar{q}_j\right)^2} \right), \tag{4}$$

where $1/(P_{ik}P_{jk})$, $1/P_{ik}$ and $1/P_{jk}$ are three coefficients employed to achieve a balance among different web services which were invoked by a user more than once.

The above Eqs. (3) and (4) often overestimate the similarity between two users who are not similar actually but happen to have similar QoS observations on a small number of co-invoked web services. To deal with this problem, we utilize a similarity weight proposed in [9], [10] to reduce the impact of a small number of co-invoked web services

$$w_{ij}'^{u} = \frac{2|S_{u_i} \cap S_{u_j}|}{|S_{u_i}| + |S_{u_j}|} w_{ij}^{u}, \tag{5}$$

where $|S_{u_i}|$ and $|S_{u_j}|$ are the numbers of web services invoked by u_i and u_j, respectively, and $|S_{u_i} \cap S_{u_j}|$ is the number of their co-invoked web services. $w_{ij}'^{u}$ ranges from -1 to 1, with a larger $w_{ij}'^{u}$ indicating a higher similarity between u_i and u_j.

Likewise, only considering the simple case that each web service was invoked by a service user at most once, the time-aware similarity between two web services s_k and s_l is defined by Eq. (6)

$$w_{kl}^{s} = \frac{\sum_{u_i \in U}(q_{ik} - \bar{q}_k)(q_{il} - \bar{q}_l) f_1(t_{ik}, t_{il}) f_2(t_{ik}, t_{il})}{\sqrt{\sum_{u_i \in U}(q_{ik} - \bar{q}_k)^2} \sqrt{\sum_{u_i \in U}(q_{il} - \bar{q}_l)^2}}, \tag{6}$$

where $U = U_{s_k} \cap U_{s_l}$ is the subset of users who invoked both s_k and s_l, and $\bar{q}_k$ and $\bar{q}_l$ are the average QoS values of s_k and s_l observed by their respective users.

Furthermore, we consider the complicated case that a web service was invoked by a service user more than once. For each common user u_i of two web services s_k and s_l, we suppose that he invoked s_k total P_{ik} times and invoked s_l total P_{il} times. Therefore, the (i,k)th and (i,l)th entries of $\boldsymbol{Q}$ can be denoted by $Q_{ik} = \{(q_{ik}^{p_k}, t_{ik}^{p_k})|1 \le p_k \le P_{ik}\}$ and $Q_{il} = \{(q_{il}^{p_l}, t_{il}^{p_l})|1 \le p_l \le P_{il}\}$, respectively. Here, $q_{ik}^{p_k}$ and $t_{ik}^{p_k}$ are the QoS value and timestamp of u_i's p_kth invocation of s_k, and $q_{il}^{p_l}$ and $t_{il}^{p_l}$ are the QoS value and timestamp of u_i's p_lth invocation of s_l. Accordingly, there are total $P_{ik}P_{il}$ pairs of $((q_{ik}^{p_k}, t_{ik}^{p_k}), (q_{il}^{p_l}, t_{il}^{p_l}))$ to be considered for service similarity computation. We denote them by $\{((q_{ik}^{(p)}, t_{ik}^{(p)}), (q_{il}^{(p)}, t_{il}^{(p)}))|(q_{ik}^{(p)}, t_{ik}^{(p)}) \in Q_{ik}, (q_{il}^{(p)}, t_{il}^{(p)}) \in Q_{il}, 1 \le p \le P_{ik}P_{il}\}$. Therefore, the time-aware service similarity computation is modified, as shown in Eq. (7):

$$w_{kl}^{s} = \sum_{u_i \in U} \left(\frac{1}{P_{ik}P_{il}} \sum_{p=1}^{P_{ik}P_{il}} \left(q_{ik}^{(p)} - \bar{q}_k\right)\left(q_{il}^{(p)} - \bar{q}_l\right) \times f_1\left(t_{ik}^{(p)}, t_{il}^{(p)}\right) f_2\left(t_{ik}^{(p)}, t_{il}^{(p)}\right) \right) \Bigg/ \left(\sqrt{\sum_{u_i \in U} \frac{1}{P_{ik}} \sum_{p_k=1}^{P_{ik}} \left(q_{ik}^{p_k} - \bar{q}_k\right)^2} \times \sqrt{\sum_{u_i \in U} \frac{1}{P_{il}} \sum_{p_l=1}^{P_{il}} \left(q_{il}^{p_l} - \bar{q}_l\right)^2} \right). \tag{7}$$

Finally, a similarity weight is employed to reduce the impact of a small number of common users:

$$w_{kl}'^{s} = \frac{2|U_{s_k} \cap U_{s_l}|}{|U_{s_k}| + |U_{s_l}|} w_{kl}^{s}, \tag{8}$$

where $|U_{s_k}|$ and $|U_{s_l}|$ are the numbers of users who invoked s_k and s_l, respectively, and $|U_{s_k} \cap U_{s_l}|$ is the number of users who invoked both s_k and s_l. $w_{kl}'^{s}$ also ranges from -1 to 1, with a larger value indicating a higher service similarity. If α and β are both set to 0, we have $f_1 = f_2 = 1$. Namely, in this particular case, the time-aware PCC degenerates into the traditional PCC described in [9], [10]. Therefore, the latter is just a special case of the former.

Algorithm 1. Time-Aware User Similarity Computation

Input:
user u_i; user u_j; time-aware user-service matrix $\boldsymbol{Q}$; current time $t_{current}$; time decay constants α, β
Output:
the time-aware similarity between u_i and u_j: $w_{ij}'^{u}$

$S_{u_i} \leftarrow \emptyset$; $S_{u_j} \leftarrow \emptyset$; $sumq_i \leftarrow 0$; $sumq_j \leftarrow 0$; $sum \leftarrow 0$;
$n \leftarrow$ the number of columns of $\boldsymbol{Q}$;
for $k \leftarrow 1$ to n **do**
  if $P_{ik} \neq 0$ **then**
    $S_{u_i} \leftarrow S_{u_i} \cup \{s_k\}$;
    for $p \leftarrow 1$ to P_{ik} **do**
      $sumq_i \leftarrow sumq_i + q_{ik}^{p}$;
    end for
  end if
  if $P_{jk} \neq 0$ **then**
    $S_{u_j} \leftarrow S_{u_j} \cup \{s_k\}$;
    for $p \leftarrow 1$ to P_{jk} **do**
      $sumq_j \leftarrow sumq_j + q_{jk}^{p}$;
    end for
  end if
end for
$\bar{q}_i \leftarrow sumq_i / \sum_{k=1}^{n} P_{ik}$; $\bar{q}_j \leftarrow sumq_j / \sum_{k=1}^{n} P_{jk}$;
$S \leftarrow S_{u_i} \cap S_{u_j}$;
$sim\_weight \leftarrow 2|S|/(|S_{u_i}| + |S_{u_j}|)$;
$A \leftarrow \sqrt{\sum_{s_k \in S} \frac{1}{P_{ik}} \sum_{p_i=1}^{P_{ik}} (q_{ik}^{p_i} - \bar{q}_i)^2} \sqrt{\sum_{s_k \in S} \frac{1}{P_{jk}} \sum_{p_j=1}^{P_{jk}} (q_{jk}^{p_j} - \bar{q}_j)^2}$;
for each service $s_k \in S$ **do**
  $sum\_es \leftarrow 0$;
  for $p \leftarrow 1$ to $P_{ik}P_{jk}$ **do**
    $f_1 \leftarrow e^{-\alpha|t_{ik}^{(p)} - t_{jk}^{(p)}|}$; $f_2 \leftarrow e^{-\beta|t_{current} - (t_{ik}^{(p)} + t_{jk}^{(p)})/2|}$;
    $sum\_es \leftarrow sum\_es + (q_{ik}^{(p)} - \bar{q}_i) \times (q_{jk}^{(p)} - \bar{q}_j) \times f_1 \times f_2$;
  end for
  $sum \leftarrow sum + \frac{1}{P_{ik}P_{jk}} \times sum\_es$;
end for
$w_{ij}'^{u} \leftarrow sim\_weight \times sum/A$;
return $w_{ij}'^{u}$;

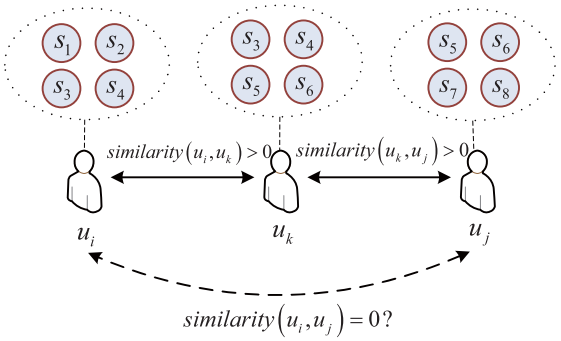


Fig. 4. Transitive similarity.

The entire procedure of the time-aware user similarity calculation is shown in Algorithm 1. Lines 1 and 2 perform some initializations. And $sumq_i$ and $sumq_j$ are the summations of QoS values observed by u_i and u_j, respectively. Lines 3-17 calculate the users' average QoS values. Then, the similarity weight and the denominator of time-aware PCC are computed (lines 18-20). Afterwards, the contribution of each co-invoked web service, which is denoted by sum_es, is added to a variable sum (lines 21-28). Finally, the similarity between u_i and u_j is calculated and returned (lines 29 and 30). The service similarity calculation is similar to Algorithm 1. Due to limited space, no more tautology here.

5 Personalized Random Walk to Handle Data Sparsity

5.1 Similarity Transition

After similarity calculation, similar neighbors should be identified for the target user and each candidate web service. Only users or services whose similarities are greater than 0 can be considered as similar neighbors [9], [10]. However, the user-service matrix is usually sparse. Even very active users may invoke just a few web services, and even very popular web services may be invoked by only a few users. Therefore, many users may have no co-invoked web service, and many web services may also have no common service user. According to Algorithm 1, the time-aware similarity between two users who have no co-invoked web service or two services which share no common user is calculated as 0. Thus, we cannot get enough similar neighbors to extract plentiful historical information for reliable and accurate QoS prediction.

However, taking user similarity as an example, no co-invoked web service does not really mean no similarity. A counter-example is illustrated in Fig. 4. Suppose that u_i invoked four web services $s_1 \sim s_4$, and u_k invoked $s_3 \sim s_6$, and u_j invoked $s_5 \sim s_8$. According to the above time-aware user similarity model, the similarity between u_i and u_j is calculated as 0 due to $|S_{u_i} \cap S_{u_j}| = 0$. However, from another point of view, u_i and u_k commonly invoked s_3 and s_4, and we assume that their calculated similarity is greater than 0, namely $w'^u_{ik} > 0$. Similarly, u_k and u_j commonly invoked s_5 and s_6, and we also assume $w'^u_{kj} > 0$. In this case, we cannot simply consider that the similarity between u_i and u_j is 0, since they have a common similar neighbor u_k and similarity is a transitive relationship. Therefore, u_i and u_j can establish an indirect similarity relationship via their common similar neighbor u_k. If two users have no co-invoked web service, but their indirect similarity is greater than 0, they are referred to as *indirect similar neighbors*.

5.2 Indirect Similarity Inference

5.2.1 Personalized Random Walk Algorithm

The random walk algorithm is similar to Google's PageRank algorithm [27], which is originally designed for web pages ranking. PageRank performs random walk on an entire web graph, which is constructed by treating web pages as nodes and the links between pages as edges. Each random step is either walking through one of the outgoing links of the current page or jumping randomly to any page with a uniform probability. Finally, the stationary visiting probability of each page p during the random walk is assigned to this page as its ranking score. In this paper, we apply the personalized random walk algorithm to both the user graph (constructed by treating users as nodes and the similarity associations between users as edges) and the service graph (constructed by treating services as nodes and the similarity associations between services as edges). This algorithm identifies indirect similar neighbors for the target user, who requests web service recommendation currently and each candidate service, thus to extract plentiful historical QoS information from these similar neighbors for more accurate QoS prediction.

Here, we utilize the user graph to detail the identification of indirect similar user neighbors. A higher score a user gets after a personalized random walk implies that (s)he is more similar to the target user. To perform personalized random walk, we first select $Top{-}\mathrm{K}$ direct similar neighbors for each user to build a user adjacency matrix:

$$\boldsymbol{W_u} = \begin{pmatrix} \hat{w}^u_{11} & \hat{w}^u_{12} & \cdots & \hat{w}^u_{1m} \\ \hat{w}^u_{21} & \hat{w}^u_{22} & \cdots & \hat{w}^u_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{w}^u_{m1} & \hat{w}^u_{m2} & \cdots & \hat{w}^u_{mm} \end{pmatrix}. \tag{9}$$

Here, if u_j $(1 \le j \le m)$ is one of the $Top{-}\mathrm{K}$ direct similar neighbors of u_k $(1 \le k \le m)$ and w'^u_{kj} (the time-aware similarity between u_k and u_j) is positive, $\hat{w}^u_{kj}$ is set equal to w'^u_{kj}, otherwise $\hat{w}^u_{kj}$ is set to 0. Additionally, $\hat{w}^u_{kk} = 0$ $(1 \le k \le m)$ means that the similarity between a user and himself is not considered.

In a user ranking (column) vector $\boldsymbol{r}$, each entry r_j $(1 \le j \le m)$ denotes the stationary visiting probability of u_j. And $\boldsymbol{r}$ is defined as the solution of the following equation:

$$\boldsymbol{r}^n = d \times \tilde{\boldsymbol{W}}_{\boldsymbol{u}} \times \boldsymbol{r}^{n-1} + (1-d) \times \boldsymbol{p}, \tag{10}$$

where $\tilde{\boldsymbol{W}}_{\boldsymbol{u}}$, the column normalized matrix of $\boldsymbol{W_u}$, is the user transition probability matrix, whose each entry $\tilde{w}^u_{kj}$ $(1 \le k, j \le m)$ represents the probability of u_k being the next state when the current state is u_j, $\boldsymbol{p}$ is a personalized column vector concentrated on a single node, namely the target user, and its each entry p_j $(1 \le j \le m)$ is defined as

$$p_j = \begin{cases} 1, & \text{if } j = i \text{ } (u_i \text{ is the target user}), \\ 0, & \text{otherwise}, \end{cases} \tag{11}$$

and $d \in (0,1)$ (A common choice for d is 0.85 [28], [29]) is a damping factor used to control the probability of the random walk restarting to the personalized vector $\boldsymbol{p}$ (with a probability $1-d$) or moving forward from the current node (with a probability d) at each iteration step. The initial value of $\boldsymbol{r}$ is defined by Eq. (12):

$$\boldsymbol{r}^0 = \frac{1}{|m|} \times \mathbf{1}_{|m|}, \tag{12}$$

where $|m|$ is the total number of service users. The definition of $\boldsymbol{r}^0$ means that at the beginning of the iteration, all users share a same ranking score. After finite iterations, $\boldsymbol{r}$ converges to a stationary distribution vector.

In an ideal situation, when the ranking vector converges, it must satisfy $\boldsymbol{r} = d \times \tilde{\boldsymbol{W}}_{\boldsymbol{u}} \times \boldsymbol{r} + (1-d) \times \boldsymbol{p}$, so we can simplify the iteration calculation by transforming the ranking vector to:

$$\boldsymbol{r} = (1-d) \times (\boldsymbol{I} - d \times \tilde{\boldsymbol{W}}_{\boldsymbol{u}})^{-1} \times \boldsymbol{p}. \tag{13}$$

Thus we can pre-compute the matrix $(1-d) \times (\boldsymbol{I} - d \times \tilde{\boldsymbol{W}}_{\boldsymbol{u}})^{-1}$, which is independent from any personalized vector $\boldsymbol{p}$. Then we can multiply the inverse matrix and the personalized vector to get the ranking vector [30].

In the converged ranking vector $\boldsymbol{r}$, each entry r_j ($1 \le j \le m, j \ne i$) is a long-term visiting probability of u_j. Therefore, r_j can be considered as a measurement of the proximity between u_j and the target user u_i, with a larger r_j indicating a closer proximity. Specially, the ith entry r_i (i is the ID of the target user) of the converged $\boldsymbol{r}$ should be set to 0, because we do not consider the proximity between the target user and himself.

5.2.2 Similarity Reconstruction

The stationary ranking vector $\boldsymbol{r}$ denotes the relative proximities rather than the actual similarities between the target user and his neighbors. The actual similarities are necessary to QoS prediction. Therefore, before QoS prediction, the actual similarities between users should be reconstructed. Suppose that the $Top{-}\mathrm{K}$ direct similar neighbors of the target user u_i is denoted by $N_{u_i} = \{u_{i_l} | 1 \le l \le K, i_l \in [1,m]\}$. And the time-aware similarity $\hat{w}^u_{ii_l}$ between u_i and u_{i_l} should be greater than 0, otherwise u_{i_l} should be removed from N_{u_i}. Finally, the reconstructed similarity (row) vector $\boldsymbol{s}_{u_i}$, which represents the actual similarities between the target user u_i and all his direct and indirect similar neighbors, is reconstructed as

$$\boldsymbol{s}_{u_i} = \frac{1}{|N_{u_i}|} \times \sum_{l=1}^{|N_{u_i}|} \frac{\hat{w}^u_{ii_l}}{r_{i_l}} \times \boldsymbol{r}^{\mathrm{T}}, \tag{14}$$

where r_{i_l} is the i_lth entry of the stationary ranking vector $\boldsymbol{r}$. We summarize the user similarity inference procedure in Algorithm 2. Line 1 builds the user adjacency matrix. Line 2 initializes the personalized vector $\boldsymbol{p}$. Lines 3-8 build the transition probability matrix by normalizing each column of the user adjacency matrix. Lines 10 and 11 compute the stationary ranking vector, by pre-computing the inverse matrix (line 9). Finally, the reconstructed similarity vector for the target user is calculated and returned (lines 12-20). Here, we only present the simplified method based on an pre-computed inverse matrix to get the final ranking vector. Of course, the iteration method can be used as an alternative. And the service similarity inference algorithm is similar to Algorithm 2.

Algorithm 2. User Similarity Inference

Input:
calculated time-aware user similarities; target user u_i, damping factor d, $Top{-}K$

Output:
reconstructed similarity vector for u_i: $\boldsymbol{s}_{u_i}$

1: build user adjacency matrix $\boldsymbol{W}_u$;
2: $\boldsymbol{p} \leftarrow [0, \ldots, 1, 0, \ldots, 0]^{\mathrm{T}}$; //only the ith entry of $\boldsymbol{p}$ is 1
3: **for** $j \leftarrow 1$ to m **do**
4: $\quad colsum \leftarrow \sum_{k=1}^{m} \hat{w}^u_{kj}$;
5: $\quad$ **for** $k \leftarrow 1$ to m **do**
6: $\quad\quad \tilde{w}^u_{kj} \leftarrow \hat{w}^u_{kj} / colsum$; //column normalization
7: $\quad$ **end for**
8: **end for**
9: $\boldsymbol{inverseMatrix} \leftarrow (1-d) \times (\boldsymbol{I} - d \times \tilde{\boldsymbol{W}}_{\boldsymbol{u}})^{-1}$;
10: $\boldsymbol{r} \leftarrow \boldsymbol{inverseMatrix} \times \boldsymbol{p}$;
11: $r_i \leftarrow 0$; //r_i is the ith entry of $\boldsymbol{r}$
12: $sum \leftarrow 0$; $N_{u_i} \leftarrow \emptyset$;
13: **for** $j \leftarrow 1$ to m **do**
14: $\quad$ **if** $\hat{w}^u_{ij} > 0$ **then**
15: $\quad\quad N_{u_i} \leftarrow N_{u_i} \cup \{u_j\}$;
16: $\quad\quad sum \leftarrow sum + \hat{w}^u_{ij} / r_j$; //$r_j$ is the j_{th} entry of $\boldsymbol{r}$
17: $\quad$ **end if**
18: **end for**
19: $\boldsymbol{s}_{u_i} \leftarrow 1/|N_{u_i}| \times sum \times \boldsymbol{r}^{\mathrm{T}}$;
20: **return** $\boldsymbol{s}_{u_i}$;

6 Time-Aware QoS Prediction

In this section, we utilize the historical QoS information from all direct and indirect similar users and similar services to make QoS prediction. Suppose that u_i and s_k are the target user and one of the candidate web services, respectively. Additionally, u_j is a similar user of u_i, and s_l is a similar service of s_k. In order to utilize time information to make more accurate QoS prediction, we define another two time factors: $f_3(t_{jk}) = e^{-\gamma_1 |t_{current} - t_{jk}|}$ and $f_4(t_{il}) = e^{-\gamma_2 |t_{current} - t_{il}|}$, where t_{jk} is the timestamp when u_j invoked s_k, t_{il} is the timestamp when u_i invoked s_l, and $\gamma_1, \gamma_2 \ge 0$ are two nonnegative decay constants. f_3 and f_4 both range from 1 to 0. The definition of f_3 means that if the timespan between u_j's invocation of s_k and the current moment is shorter, the observed value of a QoS parameter from u_j on s_k contributes more to the user-based prediction for this QoS parameter. Similarly, f_4 indicates that if the timespan between u_i's invocation of s_l and the current moment is shorter, the observed value of a QoS parameter from u_i on s_l contributes more to the service-based QoS prediction. Accordingly, the time-aware user-based and service-based QoS prediction results for the target user u_i on the candidate service s_k are respectively given by Eqs. (15) and (16):

$$\hat{q}^u_{ik} = \bar{q}_i + \frac{\sum_{u_j \in R_{u_i}} \boldsymbol{s}_{u_i}(j) f_3(t_{jk})(q_{jk} - \bar{q}_j)}{\sum_{u_j \in R_{u_i}} \boldsymbol{s}_{u_i}(j) f_3(t_{jk})}, \tag{15}$$

$$\hat{q}^s_{ik} = \bar{q}_k + \frac{\sum_{s_l \in R_{s_k}} \boldsymbol{s}_{s_k}(l) f_4(t_{il})(q_{il} - \bar{q}_l)}{\sum_{s_l \in R_{s_k}} \boldsymbol{s}_{s_k}(l) f_4(t_{il})}, \tag{16}$$

where $\boldsymbol{s}_{u_i}(j)$ is the jth entry of $\boldsymbol{s}_{u_i}$ (the reconstructed similarity vector for u_i), $\boldsymbol{s}_{s_k}(l)$ is the lth entry of $\boldsymbol{s}_{s_k}$ (the

reconstructed similarity vector for s_k), and $R_{u_i} = \{u_j | u_j \in U, \boldsymbol{s}_{u_i}(j) > 0\}$ and $R_{s_k} = \{s_l | s_l \in S, \boldsymbol{s}_{s_k}(l) > 0\}$ are the sets of all similar neighbors of u_i and s_k, respectively. If γ_1 and γ_2 are both set to 0, the time-aware QoS prediction approach degenerates into the original QoS prediction method [9], [10], so the latter is just a special case of the former.

Furthermore, considering the complicated case that a user invoked a web service more than once, the modified time-aware user-based and service-based QoS prediction results are respectively shown in Eqs. (17) and (18)

$$\hat{q}^u_{ik} = \bar{q}_i + \frac{\sum_{u_j \in R_{u_i}} \boldsymbol{s}_{u_i}(j) \left(\frac{1}{P_{jk}} \sum_{p=1}^{P_{jk}} f_3(t^p_{jk}) \left(q^p_{jk} - \bar{q}_j \right) \right)}{\sum_{u_j \in R_{u_i}} \boldsymbol{s}_{u_i}(j) \left(\frac{1}{P_{jk}} \sum_{p=1}^{P_{jk}} f_3(t^p_{jk}) \right)}, \quad (17)$$

$$\hat{q}^s_{ik} = \bar{q}_k + \frac{\sum_{s_l \in R_{s_k}} \boldsymbol{s}_{s_k}(l) \left(\frac{1}{P_{il}} \sum_{p=1}^{P_{il}} f_4(t^p_{il}) \left(q^p_{il} - \bar{q}_l \right) \right)}{\sum_{s_l \in R_{s_k}} \boldsymbol{s}_{s_k}(l) \left(\frac{1}{P_{il}} \sum_{p=1}^{P_{il}} f_4(t^p_{il}) \right)}, \quad (18)$$

where P_{jk} is the number of times that u_j invoked s_k, P_{il} is the number of times that u_i invoked s_l, q^p_{jk} and t^p_{jk} are the QoS value and timestamp of u_j's pth ($1 \le p \le P_{jk}$) invocation of s_k, and q^p_{il} and t^p_{il} are the QoS value and timestamp of u_i's pth ($1 \le p \le P_{il}$) invocation of s_l.

Finally, in order to merge the two kinds of prediction results into a consolidated prediction result, two confidence weights con_u and con_s [9], [10] are utilized to balance the user-based and service-based prediction results. They are defined as

$$con_u = \sum_{u_j \in R_{u_i}} \frac{\boldsymbol{s}_{u_i}(j)}{\sum_{u_j \in R_{u_i}} \boldsymbol{s}_{u_i}(j)} \times \boldsymbol{s}_{u_i}(j), \quad (19)$$

$$con_s = \sum_{s_l \in R_{s_k}} \frac{\boldsymbol{s}_{s_k}(l)}{\sum_{s_l \in R_{s_k}} \boldsymbol{s}_{s_k}(l)} \times \boldsymbol{s}_{s_k}(l). \quad (20)$$

Then, we can get the final prediction result by employing the following equation:

$$\hat{q}_{ik} = h_u \times \hat{q}^u_{ik} + h_s \times \hat{q}^s_{ik}, \quad (21)$$

where h_u and h_s [9], [10] are the weights of user-based and service-based prediction results and defined as

$$h_u = \frac{\lambda \times con_u}{\lambda \times con_u + (1-\lambda) \times con_s}, \quad (22)$$

$$h_s = \frac{(1-\lambda) \times con_s}{\lambda \times con_u + (1-\lambda) \times con_s}, \quad (23)$$

where λ ($0 \le \lambda \le 1$) is utilized to determine how much the final prediction result relies on the user-based and the service-based prediction results. For more details about con_u, con_s and λ, please refer to [9], [10].

When $R_{u_i} \neq \emptyset \wedge R_{s_k} = \emptyset$, namely no similar web service, the missing QoS value prediction relies only on similar users by utilizing Eq. (15) or (17). Similarly, when $R_{u_i} = \emptyset \wedge R_{s_k} \neq \emptyset$, the QoS prediction degrades to the service-based method by utilizing Eq. (16) or (18). When $R_{u_i} = \emptyset \wedge R_{s_k} = \emptyset$, since there is no similar user nor similar web service, the prediction $\hat{q}_{ik}$ is just defined as

$$\hat{q}_{ik} = \frac{1}{2} \times (\bar{q}_i + \bar{q}_k), \quad (24)$$

where $\bar{q}_i$ is the average QoS value of different web services invoked by u_i, and $\bar{q}_k$ is the average QoS value of s_k observed by different users.

After predicting the missing values on multiple QoS properties, web service recommendation can be made. The service recommender system evaluates each candidate web service by applying a Multiple Criteria Decision Making (MCDM) [19] technique. Then, the candidate web service which has optimal comprehensive QoS performance will be recommended to the target user.

7 Computational Complexity Analysis

In this section, we discuss the computational complexity of our web service recommendation algorithm. Here, we only analyse the computational complexity of the simple case that a user invoked a web service at most once. Under the background of data sparsity, a user rarely invoked a web service more than once. Even if there exist some special cases, we think that they would not affect the computational complexity significantly.

7.1 Complexity of Similarity Computation

In Section 4, the computational complexity of time-aware user similarity computation based on Eq. (3) or (4) is $O(n)$, since there are at most n co-invoked web services between two users. Similarly, the computational complexity of time-aware service similarity computation based on Eq. (6) or (7) is $O(m)$, since there are at most m common users between two web services. As shown in Algorithm 2, before similarity inference, all users' and services' similarities should be computed. Hence, we need to compute similarities for total m^2 pairs of users, so the computational complexity of this procedure is $O(m^2 n)$. Analogously, the computational complexity of similarity computation for all services is $O(mn^2)$.

7.2 Complexity of Indirect Similarity Inference

Similarity inference totally includes four steps: constructing the transition probability matrix, pre-computing the inverse matrix, performing personalized random walk and similarity reconstruction. 1) The transition probability matrix can be built by normalizing each column of the adjacency matrix. The computational complexity of column normalization of the user adjacency matrix is $O(m^2)$, since the normalization of each column needs $(m-1)$ additions and m divisions and total m columns need to be normalized. Likewise, the complexity of building a service transition probability matrix is $O(n^2)$. 2) Pre-computing the inverse matrix has time complexity $O(m^3)$ for users and $O(n^3)$ for services. Tong et al. [31] has shown that we can calculate the inverse matrix with reasonable cost by using dimension reduction techeniques. And some clustering methods are presented in [32] to reduce its computational cost. 3) Based on the pre-computed inverse matrix, performing personalized random walk for each user has time complexity $O(m)$, since we need to multiply the $m \times m$ inverse matrix with

TABLE 1
Computational Complexity of Different Procedures

Procedure	Complexity	Procedure	Complexity	Procedure	Complexity
Each user similarity computation	$O(n)$	Each service similarity computation	$O(m)$	Similarity inference for each user	$O(m^3)$
All users' similarities computation	$O(m^2n)$	All services' similarities computation	$O(mn^2)$	Similarity inference for each service	$O(n^3)$
User transition probability matrix construction	$O(m^2)$	Service transition probability matrix construction	$O(n^2)$	User-based QoS prediction for a target user	$O(mn)$
User inverse matrix computation	$O(m^3)$	Service inverse matrix computation	$O(n^3)$	Service-based QoS prediction for a target user	$O(n^2)$
Personalized random walk for each user	$O(m)$	Personalized random walk for each service	$O(n)$	Hybrid QoS prediction for a target user	$O(mn+n^2)$
Similarity reconstruction for each user	$O(m)$	Similarity reconstruction for each service	$O(n)$	Time-aware and data sparsity tolerant Web service recommendation for a target user	$O(m^2n+mn^2+m^3+n^3)$

each $m \times 1$ personalized vector, and each personalized vector contains only one non-zero entry and all other zero entries. Similarly, the complexity of personalized random walk for each service is $O(n)$. 4) The computational complexity of similarity reconstruction for the target user is $O(m)$, since at most m similarities need to be reconstructed. Analogously, the complexity of similarity reconstruction for each candidate web service is $O(n)$, since at most n similarities need to be reconstructed.

Accordingly, the complexity of indirect similarity inference is a linear combination of the four steps, namely $O(m^3)$ for each user and $O(n^3)$ for each service.

7.3 Complexity of QoS Prediction for a Target User

The computational complexity of user-based prediction of each missing QoS value for the target user is $O(m)$, since at most m similar users will be utilized for QoS prediction. And there are at most n missing QoS values to be predicted for a target user, so the complexity of user-based QoS prediction for a target user is $O(mn)$. The complexity of the service-based prediction for the target user on each candidate web service is $O(n)$, since no more than n similar web services are used for QoS prediction. Similarly, there are at most n missing values to be predicted for the target user, so the complexity of service-based QoS prediction for a target user is $O(n^2)$. Since the hybrid QoS prediction for a target user is a linear combination of the user-based prediction and service-based prediction, its complexity is $O(mn+n^2)$.

7.4 Complexity of Time-Aware and Data Sparsity Tolerant Web Service Recommendation for A Target User

The proposed time-aware and data sparsity tolerant web service recommendation algorithm for a target user includes the similarity computations for all users ($O(m^2n)$) and all services ($O(mn^2)$), indirect similarity inference for the target user ($O(m^3)$) and at most n candidate web services ($O(n^3)$) and the hybrid QoS prediction for a target user ($O(mn+n^2)$). Therefore, the total complexity of web service recommendation for a target user is the a linear combination of these procedures, namely $O(m^2n+mn^2+m^3+n^3)$.

We can observe that the computational complexities of the similarities computation for all users and services and pre-computing the inverse matrices are relatively high. However, they are acceptable because these procedures only need to be pre-performed once and then can be shared by all users and services for web service recommendation. Finally, all computational complexities analysed above are summarized in Table 1.

8 Experiments

In this section, several experiments are conducted on a real-world dataset to validate the effectiveness of our approach.

8.1 Dataset

We use a freely available dataset provided by Zhang et al. [33] for experiments. They employed 142 distributed computers from Planet-Lab[3] to evaluate the *response time* and *throughput* of 4,532 web services during 64 continuous time intervals, with each time interval lasting for 15 minutes. Accordingly, the dataset contains 64 matrices for *response time* and 64 matrices for *throughput*, and each matrix has 142 rows and 4,532 columns.

8.2 Data Processing

Before experimenting, we process the raw data so that it can be used to verify the effectiveness of our approach. We take the data processing on *response time* for example, and that on *throughput* is similar. We randomly select 140 users and 140 services from the dataset to build 64 user-service *response time* matrices, with each time interval providing one 140×140 matrix. Some initially selected web services should be replaced by others due to invocation failure. Then, we randomly divide the matrix of the 64th (latest) time interval into two parts, one as the training matrix, which contains about 60 percent data of the whole matrix, and the other as the test matrix, which contains the remaining 40 percent data. Afterwards, we randomly select 90 percent entries of the training matrix and replace each of them with the corresponding QoS value observed during any of the previous 63 time intervals. Thus, a new training matrix is constructed, which contains training data from different time intervals. The 64th time interval is seen as the current moment (the recommendation time). Since we do not need

3. http://www.planet-lab.org

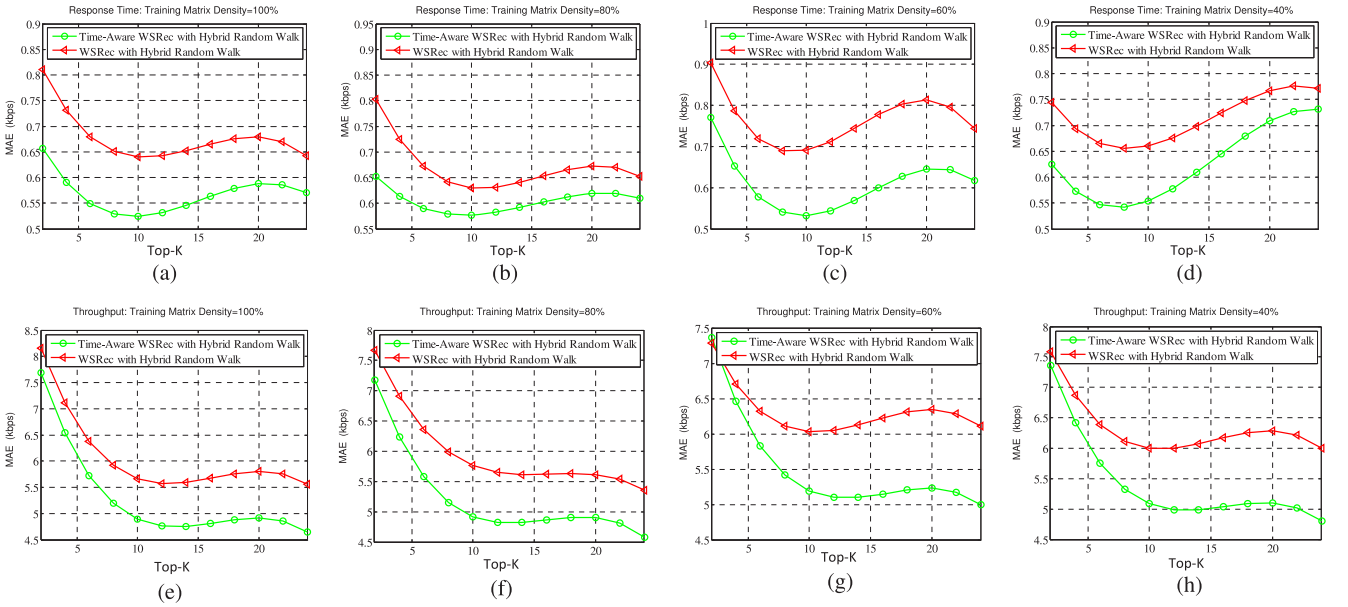


Fig. 5. Evaluation of time-aware approach.

to predict missing QoS values before the current time, the test matrix only contains the QoS data observed at the 64th time interval.

8.3 Evaluation Metric

We use Mean Absolute Error (MAE) [34] to measure the prediction accuracy of our approach in comparison with other methods. MAE is a metric widely used to measure the prediction accuracy of CF methods, and defined as

$$MAE = \frac{1}{N} \times \sum_{i,k} |\hat{q}_{ik} - q_{ik}|, \tag{25}$$

where $\hat{q}_{ik}$ and q_{ik} are the predicted and actual QoS values of s_k observed by u_i, respectively, and N is the total number of predicted values. A smaller MAE indicates a more accurate QoS prediction.

8.4 Performance Comparison

First, we randomly remove some entries of the training matrix to make training data density vary from 100 to 40 percent, with a step value of -20 percent, to simulate different sparsity levels of training data. The remaining training data are used for time-aware similarity calculation. Then, we choose the $Top{-}\mathrm{K}$ most similar neighbors for each user or service as its direct similar neighbors. A similar neighbor should be removed from the $Top{-}\mathrm{K}$ similar neighbors if its similarity is equal to or smaller than 0. For the sake of simplicity, the values of $Top{-}\mathrm{K}$ for each user and each service are set equal, simultaneously varying from 2 to 24, with a step value of 2. The value of λ is fixed at 0.5. Additionally, the four time decay constants α, β, γ_1 and γ_2 are also set equal.

We utilize the hybrid CF named "WSRec" [9], [10] as the baseline QoS prediction algorithm. If "WSRec" performs random walk while seeking for indirect similar users and services, it is named "WSRec with Hybrid Random Walk". If "WSRec" not only performs hybrid random walk, but also integrates time information into similarity measurement and QoS prediction, it is named "Time-Aware WSRec with Hybrid Random Walk", namely the approach proposed in this paper.

We first validate the effectiveness of the time-aware approach, which integrates time information into both similarity computation and QoS prediction. We compare the MAE of "WSRec with Hybrid Random Walk" with that of "Time-Aware WSRec with Hybrid Random Walk" on both the *response time* and *throughput* datasets. Each experiment loops 50 times and the average values are reported. Figs. 5a to 5d show the performance comparison of the two approaches under different training data densities of *response time*, and Figs. 5e to 5h show their performance comparison on *throughput*. It can be observed that "Time-Aware WSRec with Hybrid Random Walk" outperforms "WSRec with Hybrid Random Walk" obviously. This indicates that the time information is valuable to QoS prediction. Here, the duration of each time interval (15 minutes) is simply denoted as 1 and the time decay constants α, β, γ_1 and γ_2 are all set to 0.085 for *response time* or 0.09 for *throughput* due to the optimal performance the two values yield.

Next, we corroborate the effectiveness of the hybrid personalized random walk algorithm for alleviating data sparsity, by comparing the performance of the four algorithms: "Time-Aware WSRec without Random Walk", "Time-Aware WSRec with User Random Walk", "Time-Aware WSRec with Service Random Walk" and "Time-Aware WSRec with Hybrid Random Walk". The first algorithm is the basic "WSRec" introducing time information into similarity calculation and QoS prediction. The second algorithm is based on the first one, and additionally performs personalized random walk while discovering indirect similar neighbors for the target user. Similarly, the third algorithm is also based on the first one and performs personalized random walk to identify indirect similar neighbors for each candidate web service. The last one is the combination of the second and the third algorithms. Each experiment loops 50 times and we report the average values. The performance comparisons of the four algorithms under different training

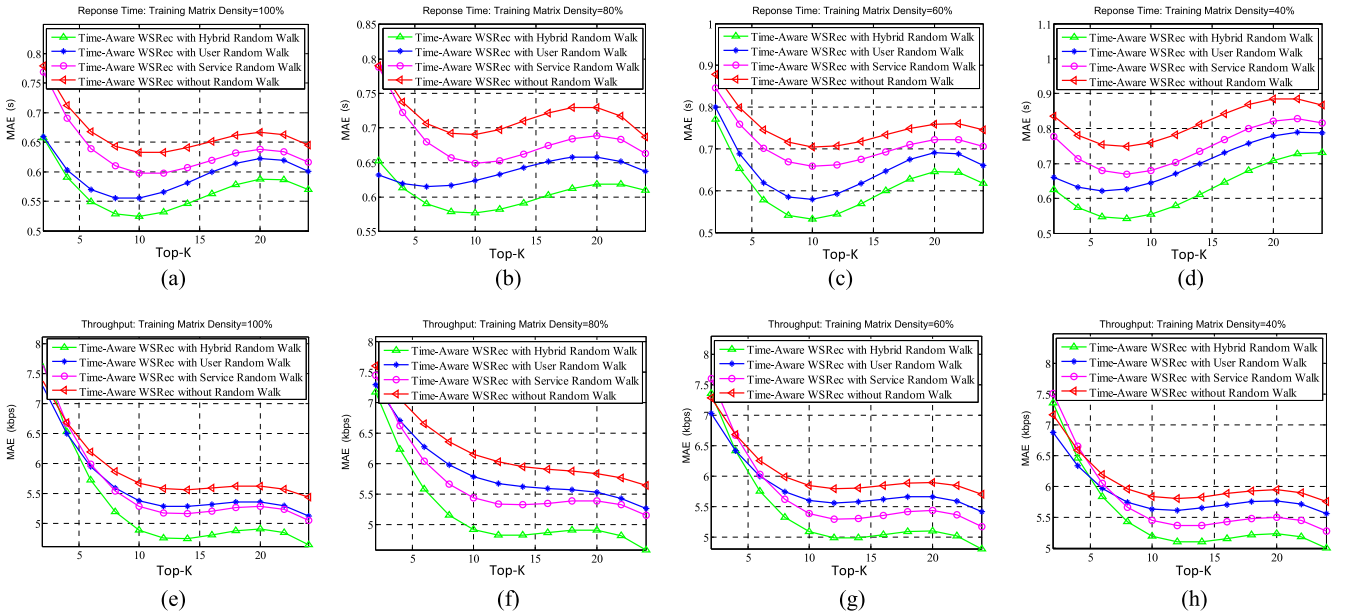


Fig. 6. Evaluation of personalized random walk approach.

data densities of *response time* and *throughput* are respectively shown in Figs. 6a to 6d and Figs. 6e to 6h.

We can observe that under different training data densities (from 100 to 40 percent), the algorithms performing user or service personalized random walk outperform that without random walk. And as described in Section 5, the hybrid personalized random walk is the combination of user personalized random walk and service personalized random walk, so it deserves to achieve better performance than the latter two algorithms. Figs. 6a to 6h all validate this point, because the MAE of the hybrid personalized random walk is obviously smaller than that of the user or service personalized random walk. Besides, even if the training data is very sparse (40 percent), the hybrid personalized random walk algorithm can still improve the prediction accuracy remarkably compared with the algorithm without random walk, as illustrated in Figs. 6d and 6h. This shows the powerful capability of the hybrid personalized random walk algorithm to alleviate data sparsity.

8.5 The Impact of Time Decay Constants

As described in Sections 4 and 6, we employ total four time decay constants and set them equal. In this section, we investigate the influence of different decay constant values on QoS prediction accuracy. Fig. 7a shows a series of exponential decay functions, with the decay constants λ varying from 0 to 0.4 and a corresponding step value of 0.01. Its x-axis contains 64 time intervals. The shape of the function curve changes violently when λ ranges from 0 to 0.3. But when λ is greater than 0.3, the curve shape changes little. Therefore, we investigate the performance variation with the decay constants ranging from 0 to 0.3, with a step value of 0.005. Fig. 7b shows the average MAE reduction of "Time-Aware WSRec with Hybrid Random Walk" compared with "WSRec with Hybrid random walk" with different decay constant values on *response time*, and Fig. 7c shows that on *throughput*. A larger MAE reduction means a higher performance improvement. When decay constants are 0, there is no performance improvement (MAE reduction is 0). In Fig. 7b, when decay constants range from 0 to 0.085, the MAE reduction of the time-aware approach climbs quickly. When decay constants are 0.085, the MAE reduction reaches its peak. And when decay constants are greater than 0.085, the curve descends slowly. Similarly, in Fig. 7c, 0.09 is the peak value of time decay constants on the *throughput* dataset. This is why we choose 0.085 and 0.09 as the time decay constant values for *response time* and *throughput* in Section 8.4. Consequently, choosing a proper value

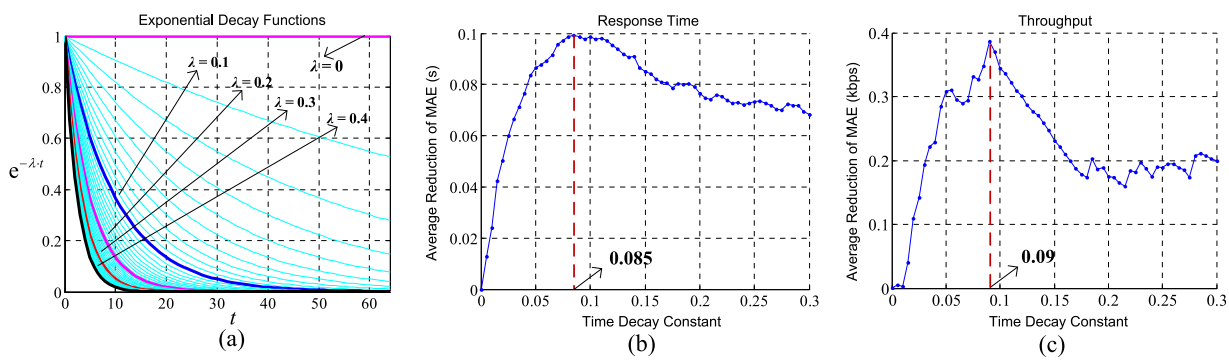


Fig. 7. Impact of different time decay constant values.

for time decay constants is conducive to getting satisfied system performance.

9 Conclusions

In this paper, we propose an improved neighborhood-based CF approach to make more accurate QoS prediction, thus to provide higher-quality web service recommendation to service users. It incorporates time information into both the similarity computations and QoS predictions to improve QoS prediction accuracy, and alleviates data sparsity effectively by performing the hybrid personalized random walk mechanism on both the user graph and the service graph.

Importantly, our approach can apply to different QoS properties, thus to provide data support for evaluating each candidate web service and generating service recommendation results by utilizing Multiple Criteria Decision Making techniques. The effectiveness and feasibility of our approach are validated by experiments.

However, the work presented in this paper still has some limitations. We do not take the properties of timestamps into consideration (e.g., is Monday or Tuesday afternoon really different from Sunday afternoon? Is Monday this week really different from last Monday?). Additionally, time needs to be regarded from a broader perspective and associated to its location to have meaning. And we lack effective mechanisms to incorporate the time dimension into a generalized context-aware technical framework for web service recommendation. All these problems are important and interesting. Therefore, in our future work, we will try to solve these problems to further improve web service recommendation performance.

Acknowledgments

The authors appreciate the reviewers for their valuable comments and suggestions for the improvement of this paper. The work described in this paper was fully supported by the Chinese 863 project (Project No. 2012AA011206).

References

[1] L. J. Zhang, J. Zhang, and H. Cai, *Services Computing*. New York, NY, USA: Springer, 2007.

[2] L. Kuang, Y. Xia, and Y. Mao, "Personalized services recommendation based on context-aware QoS prediction," in *Proc. 19th Int. Conf. Web Services*, 2012, pp. 400–406.

[3] L. Yao, Q. Z. Sheng, A. Segev, and J. Yu, "Recommending web services via combining collaborative filtering with content-based features," in *Proc. 20th Int. Conf. Web Services*, 2013, pp. 42–49.

[4] J. Huang and C. Lin, "Agent-based green web service selection and dynamic speed scaling," in *Proc. 20th Int. Conf. Web Services*, 2013, pp. 91–98.

[5] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proc. 14th Conf. Uncertainty Artif. Intell.*, 1998, pp. 43–52.

[6] R. Jin, J. Y. Chai, and L. Si, "An automatic weighting scheme for collaborative filtering," in *Proc. 27th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2004, pp. 337–344.

[7] M. Deshpande and G. Karypis, "Item-based top-n recommendation algorithms," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 143–177, Jan. 2004.

[8] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proc. 10th Int. Conf. World Wide Web*, 2001, pp. 285–295.

[9] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "Wsrec: A collaborative filtering based web service recommender system," in *Proc. 7th Int. Conf. Web Services*, 2009, pp. 437–444.

[10] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "QoS-aware web service recommendation by collaborative filtering," *IEEE Trans. Services Comput.*, vol. 4, no. 2, pp. 140–152, Apr./Jun. 2011.

[11] M. Tang, Y. Jiang, J. Liu, and X. Liu, "Location-aware collaborative filtering for QoS-based service recommendation," in *Proc. 19th Int. Conf. Web Services*, 2012, pp. 202–209.

[12] Y. Shen, J. Zhu, X. Wang, L. Cai, X. Yang, and B. Zhou, "Geographic location-based network-aware QoS prediction for service composition," in *Proc. 20th Int. Conf. Web Services*, 2013, pp. 66–74.

[13] X. Chen, Z. Zheng, Q. Yu, and M. Lyu, "Web service recommendation via exploiting location and QoS information," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 7, pp. 1913–1924, Jul. 2014.

[14] M. Papagelis, D. Plexousakis, and T. Kutsuras, "Alleviating the sparsity problem of collaborative filtering using trust inferences," in *Trust Management*. Berlin, Germany: Springer-Verlag, 2005, pp. 224–239.

[15] H. Yildirim and M. S. Krishnamoorthy, "A random walk method for alleviating the sparsity problem in collaborative filtering," in *Proc. 2nd ACM Conf. Recommender Syst.*, 2008, pp. 131–138.

[16] S. Shang, S. R. Kulkarni, P. W. Cuff, and P. Hui, "A randomwalk based model incorporating social information for recommendations," in *Proc. 22nd Int. Workshop Web Services*, 2012, pp. 1–6.

[17] Z. Zhang, D. D. Zeng, A. Abbasi, J. Peng, and X. Zheng, "A random walk model for item recommendation in social tagging systems," *ACM Trans. Manag. Inf. Syst.*, vol. 4, no. 2, p. 8, Aug. 2013.

[18] Y. Zhou, L. Liu, C.-S. Perng, A. Sailer, I. Silva-Lepe, and Z. Su, "Ranking services by service network structure and service attributes," in *Proc. 20th Int. Conf. Web Services*, 2013, pp. 26–33.

[19] N. Fakhfakh, H. Verjus, F. Pourraz, and P. Moreaux, "QoS aggregation for service orchestrations based on workflow pattern rules and MCDM method: Evaluation at design time and runtime," *Service Oriented Comput. Appl.*, vol. 7, no. 1, pp. 15–31, 2013.

[20] Y. Jiang, J. Liu, M. Tang, and X. Liu, "An effective web service recommendation method based on personalized collaborative filtering," in *Proc. 9th Int. Conf. Web Services*, 2011, pp. 211–218.

[21] X. Chen, Z. Zheng, X. Liu, Z. Huang, and H. Sun, "Personalized Qos-aware web service recommendation and visualization," *IEEE Trans. Services Comput.*, vol. 6, no. 1, pp. 35–47, Jan./Mar. 2013.

[22] H. J. Ahn, "A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem," *Inf. Sci.*, vol. 178, no. 1, pp. 37–51, 2008.

[23] C.-F. Tsai and C. Hung, "Cluster ensembles in collaborative filtering recommendation," *Appl. Soft. Comput.*, vol. 12, no. 4, pp. 1417–1425, 2012.

[24] T. Hofmann, "Latent semantic models for collaborative filtering," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 89–115, Jan. 2004.

[25] J. Canny, "Collaborative filtering with privacy via factor analysis," in *Proc. 25th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2002, pp. 238–245.

[26] M. Jiang, P. Cui, F. Wang, Q. Yang, W. Zhu, and S. Yang, "Social recommendation across multiple relational domains," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manag.*, 2012, pp. 1422–1431.

[27] A. N. Langville and C. D. Meyer, *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton, NJ, USA: Princeton Univ. Press, 2011.

[28] M. Bianchini, M. Gori, and F. Scarselli, "Inside pagerank," *ACM Trans. Internet Technol.*, vol. 5, no. 1, pp. 92–128, Feb. 2005.

[29] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna, "The query-flow graph: Model and applications," in *Proc. 17th ACM Conf. Inf. Knowl. Manag.*, 2008, pp. 609–618.

[30] S. Lee, S.-I. Song, M. Kahng, D. Lee, and S.-G. Lee, "Random walk based entity ranking on graph for multidimensional recommendation," in *Proc. 5th ACM Conf. Recommender Syst.*, 2011, pp. 93–100.

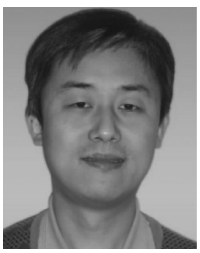
[31] H. Tong, C. Faloutsos, and J.-Y. Pan, "Random walk with restart: Fast solutions and applications," *Knowl. Inf. Syst.*, vol. 14, no. 3, pp. 327–346, Jul. 2007.

[32] H. Cheng, P.-N. Tan, J. Sticklen, and W. F. Punch, "Recommendation via query centered random walk on k-partite graph," in *Proc. IEEE 7th Int. Conf. Data Mining*, 2007, pp. 457–462.

[33] Y. Zhang, Z. Zheng, and M. R. Lyu, "WSPred: A time-aware personalized QoS prediction framework for web services," in *Proc. 22nd IEEE Int. Symp. Softw. Rel. Eng.*, 2011, pp. 210–219.

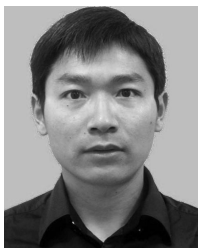
[34] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," *Climate Res.*, vol. 30, no. 1, pp. 79–82, Dec. 2005.

Yan Hu received the BEng degree in automation from Xi'an Jiaotong University, Xi'an, China, in 2011. She is currently working toward the PhD degree in the Science and Technology on Integrated Information System Laboratory, Institute of Software, Chinese Academy of Sciences and University. Her main research interests include Web service computing and information system integration.



Qimin Peng received the PhD degree in computer science from the Beijing Institute of Technology, Beijing, China, in 2005. He is currently an associate professor and a master supervisor in the Science and Technology on Integrated Information System Laboratory, Institute of Software, Chinese Academy of Sciences. His main research interests are in Web service computing, information system integration, image processing and pattern recognition.

Xiaohui Hu received the BEng degree from the Dalian University of Technology, Dalian, China, in 1982, the MEng degree from the National University of Defense Technology, Changsha, China, in 1985, and the PhD degree from the Beihang University, Beijing, China, in 2003. He is currently a professor, PhD supervisor and an assistant chief engineer in the Institute of Software, Chinese Academy of Sciences. He is also a director of the Science and Technology on Integrated Information System Laboratory. He was with the Beijing Institute of Systems Engineering as an associate professor and deputy director from 1985 to 1997. From 1997 to 2005, he was a professor of the Beijing Institute of Systems Engineering and the director of its simulation laboratory. From 2005 to 2007, he was with the Institute of Software, Chinese Academy of Sciences. His research interests include Web service computing, system simulation and information system integration.



Rong Yang received the BEng degree from the College of Computer Science at South-Central University for Nationalities in 2002 and the MEng degree from the College of Mathematics and Computer Science at Wuhan Textile University in 2009. He is currently working toward the PhD degree in State Key Lab of Software Engineering, School of Computer, Wuhan University. His main research interests include Web service computing and software engineering.