SOCA (2017) 11:33–45
DOI 10.1007/s11761-016-0191-8

ORIGINAL RESEARCH PAPER

CluCF: a clustering CF algorithm to address data sparsity problem

Chengyuan Yu[1] · **Linpeng Huang**[1]

Received: 14 November 2014 / Revised: 16 November 2015 / Accepted: 8 February 2016 / Published online: 23 February 2016

Abstract In QoS-based Web service recommendation, predicting Quality of Service (QoS) for users will greatly aid service selection and discovery. Collaborative filtering (CF) is an effective method for Web service selection and recommendation. Data sparsity is an important challenges for CF algorithms. Although model-based algorithms can address the data sparsity problem, those models are often time-consuming to build and update. Thus, these CF algorithms aren't fit for highly dynamic and large-scale environments, such as Web service recommendation systems. In order to overcome this drawback, this paper proposes a novel approach CluCF, which employs user clusters and service clusters to address the data sparsity problem and classifies the new user (the new service) by location factor to lower the time complexity of updating clusters. Additionally, in order to improve the prediction accuracy, CluCF employs time factor. Time-aware user-service matrix Mu;s(tk, d) is introduced, and the time-aware similarity measurement and time-aware QoS prediction are employed in this paper. Since the QoS performance of Web services is highly related to invocation time due to some time-varying factors (e.g., service status, network condition), time-aware similarity measurement and time-aware QoS prediction are more trustworthy than traditional similarity measurement and QoS prediction, respectively. Since similarity measurement and QoS prediction are two key steps of neighborhood-based CF, time-aware CF will be more accurate than traditional CF. Moreover, our approach systematically combines user-based and item-based methods and employs influence weights to balance these two predicted values, automatically. To validate our algorithm, this paper conducts a series of large-scale experiments based on a real-world Web service QoS dataset. Experimental results show that our approach is capable of alleviating the data sparsity problem.

Keywords Web service · QoS prediction · Time-aware · Collaborative filtering algorithm

1 Introduction

With the exponential growth of Web services, there are many Web services with identical or similar functionalities, but different QoS [1]. Thus, Web services with identical or similar functionalities would be identified by QoS which has become a critical issue in services computing [2].

The QoS performance of Web services observed from the users' perspective is usually quite different from what is declared by the service providers in SLA, mainly due to the following reasons: (1) QoS performance of Web services is highly related to invocation time, since the service status (e.g., workload, number of clients) and the network environment (e.g., congestion) change over time. For example, in throughput dataset 3,[1] which was published in reference [3] when user 141 invokes service 4499 at time interval 20, the throughput is 39.953053. But when the same user invokes the same service at different time interval 45, the throughput becomes 6.022647. The first throughput is six times larger than the second. (2) the QoS values are often different from the QoS values it declared, since Web services are in essence loosely coupled, hosted by different providers and located in

✉ Chengyuan Yu
ycy8525@sjtu.edu.cn

Linpeng Huang
huang-lp@cs.sjtu.edu.cn

[1] The Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China

[1] http://www.wsdream.net/wsdream/dataset.html.

different location and may subject to different development, verification, or testing processes.

Thus, predicting the QoS value is becoming more and more essential for service-oriented application designers to make service selection and service composition.

CF [4–7] is an effective method for Web service selection and recommendation. And CF has been widely used in commercial recommendation systems [8]. CF would automatically predict the QoS values of a target Web service for an active service user by employing historical QoS information from other similar service users, who have similar historical QoS values on the service set, in which every service is similar to the target service.

Collaborative filtering algorithms could be divided into two main categories: memory-based and model-based algorithms [9]. Memory-based CF algorithms use the entire or a sample of the user-service database to generate a prediction. By identifying neighbors of an active user, a prediction of QoS values on a target service for the active user can be produced. Model-based CF algorithms make prediction by models.

Memory-based CF algorithms are easy to implement and highly effective, but they suffer from two fundamental problems: sparsity and scalability. Sparsity refers to the fact that most users do not invoke most services and hence a very sparse user-service rating matrix is generated. As a result, the accuracy of the method will be poor. For example, Titan [10] has about 16,000 Web services. In this case, even if the average number of Web services which had been invoked by each user, up to 1,000, the data density of user-service matrix is merely 6.3 %. In this case, the accuracy of the state of the art memory-based CF algorithms will be poor, since memory-based CF algorithms don't have enough information to find enough similar users or similar services. Even sometimes, the memory-based CF algorithm will fail in this case. In scalability aspect, it suffers from serious scalability problems in failing to scale up its computation with the growth of both the number of users and the number of items. For example, Amazon.com has more than 29 million customers and several million products. The time complexity of general memory-based collaborative filtering algorithms is $O(M \times N)$, where N is the number of users and M is the number of product catalog items, since it examines N customers and up to M items for each customer. Thus, the memory-based CF algorithms encounter severe performance and scaling issues. Some model-based CF algorithms would address the sparsity and scalability problem, but those models are often time-consuming to build and update. Please refer to the "Related work" for detail.

This paper proposes a clustering CF algorithm to address the data sparsity and scalability problems.

Clustering CF algorithms address the scalability problem by seeking users for recommendation within smaller and highly similar clusters instead of the entire database, but there are trade-offs between scalability and prediction performance [9]. Our approach is a clustering CF approach. In order to improve the prediction accuracy, our approach employs time factor, since QoS performance of Web services is highly related to invocation time (QoS performance of Web services is highly related to invocation time, since the service status (e.g., workload, number of clients) and the network environment (e.g., congestion) change over time). Moreover, our approach systematically combines user-based and item-based methods to predict the missing QoS values and employs influence weights, inf_u and inf_s, to balance these two predicted values, automatically.

To alleviate the data sparsity problem, our approach improves matrix density by converting the user-service-time tensor into a user-service matrix and then converting the user-service matrix into userCluster-service matrix and user-serviceCluster matrix. For example, we suppose the number of users is 100 and the number of services is also 100, the data density of user-service matrix is 6.3 %. In this case, if the number of service clusters is about 20, then the average data density of user-serviceCluster matrix is about 27.8 %. Similarly, if the number of user clusters is about 20, then the average data density of userCluster-service matrix is also about 27.8 %. This example shows that our approach would improve the data density effectively. Generally, if the data density is larger than 20 %, the prediction accuracy of CF algorithm would be fairly higher and this forms a solid basis for our approach. According to the analysis in Sect. 3.9, the time complexity of our approach is about O(5*20) which is far smaller than the complexity of traditional memory-based CF which is O(100*100). According to the analysis in Sect. 3.2.1, the average time complexity of updating is just about O(5).

According to the analysis in Sect. 3.9, this complexity analysis shows that our approach is very efficient and can be applied to large-scale systems.

Experimental results show that our approach is capable of alleviating the data sparsity problem.

The rest of this paper is organized as follows: Sect. 2 introduces related work. Section 3 describes the novel clustering CF algorithm. Section 4 presents experiments and results. Section 5 concludes this paper.

2 Related work

In this section, we briefly present some of the research literatures related to collaborative filtering, recommender systems, and QoS prediction. Researchers have devised a number of collaborative filtering algorithms which could be divided into two main categories: memory-based and model-based algorithms [11].

Memory-based CF algorithms are easy to implement and highly effective, but they suffer from two fundamental problems: sparsity and scalability. Model-based CF algorithms can address the scalability and sparsity problems. However, those models are often time-consuming to build and update. They can be built off-line over a matter of hours or days.

To alleviate the data sparsity problem, many approaches have been proposed. Dimensionality reduction techniques, such as singular value decomposition (SVD) [12], remove unrepresentative or insignificant users or items to reduce the dimensionality of the user-item matrix directly, but have to undergo expensive matrix factorization steps. The time complexity of SVD is $O(N^3 + M^3)$ [13], where N is the number of users and M is the number of services. The time complexity of SVD-updating is $O(f^2M + f^2N)$, where f is the number of factors [14]. Thus, these CF algorithms aren't fit for highly dynamic and large-scale environments, such as Web service recommendation systems. The patented latent semantic indexing (LSI) techniques, such as Sarwar et al. [15], are based on SVD. LSI captures the similarity among users and items in a reduced dimensional space. LSI also has to undergo expensive matrix factorization steps. Contend-based filter techniques, such as GroupLens [16] and content-based collaborative recommendation [17], are found helpful to address the sparsity problem, in which external content information can be used to produce predictions for new users or new items. They recommend items based solely on a profile built up by analyzing the content of items that a user has rated. But content-based CF have difficulty in distinguishing between high-quality and low-quality information that is on the same topic. And as the number of items grows, the number of items in the same content-based category increases, further decreasing the effectiveness of content-based approaches [18]. Therefore, those approaches aren't fit for the environments with a large amount of users and services.

In order to overcome the drawbacks described above, our approach CluCF improves matrix density to alleviate the data sparsity problem. According to the analysis in Sect. 3.9 and the experiment results in Sect. 4.3, the advantages of our approach are that the models would be updated quickly which assure that our approach are fit for highly dynamic environments with massive users and services, and the prediction accuracy is fairly high when the user-service-time rating tensor is very sparse.

Clustering CF algorithms would address the scalability problem, but there are trade-offs between scalability and prediction performance. Much research effort focuses on improving the prediction accuracy. For example, Zheng et al. propose a hybrid method, which is a user-based and item-based collaborative filtering algorithm to make QoS prediction [19]. In order to further improve the prediction accuracy, different kinds of factors are taken into account when the missing QoS values are predicted by collaborative filtering algorithm. Jiang et al. [20] take into account the influence of personalization of Web service items when computing degree of similarity between users. Zhang et al. [21] take the environment factor and user input factor into account, where environment factor refers to those environmental features which have an effect on the QoS of Web service, e.g., bandwidth, and input factor refers to those input features which have an effect on QoS of Web service, e.g., input data size. Chen et al. [22] and Tang et al. [23] take location factor into account. All above works focus on improving the prediction accuracy of CF algorithms, but none of them focus on sparsity.

Our approach employs time factor to improve the prediction accuracy. Time factor is a very important factor to predict QoS, since QoS performance of Web services is highly related to invocation time. The reason why QoS performance of Web services is highly related to invocation time is that the service status (e.g., workload, number of clients) and the network environment (e.g., congestion) change over time. Moreover, our approach systematically combines user-based and item-based methods to predict the missing QoS values and employs influence weights, inf_u and inf_s, to balance these two predicted values, automatically.

3 The QoS prediction algorithm

3.1 Notations and definitions

The following are important notations used in the rest of this paper:

$U = \{u_1, u_2, \ldots, u_n\}$ is a set of service users, where n refers to the total number of service users registered in the recommendation system.

$S = \{s_1, s_2, \ldots, s_m\}$ is a set of Web services, where m refers to the total number of Web services collected by the recommendation system.

$U_c = \{uc_1, uc_2, \ldots, uc_{k'}\}$ is a set of user clusters. In other words, U_c is a partition of users set U. uc_i is a user cluster.

$S_c = \{sc_1, sc_2, \ldots, sc_{k''}\}$ is a set of service clusters. In other words, S_c is a partition of services set S. sc_i is a service cluster;

$T = \{t_1, t_2, \ldots, t_r\}$ is a set of time interval, where r refers to the total number of time intervals. For example, 1 day has 24 h with a time interval lasting for 15 min, then $r = 96$.

$M_{u,s,t} = \{q_{u_i,s_j,t_k} | 1 \le i \le n, 1 \le j \le m, 1 \le k \le r\}$ is the user-service-time tensor, where q_{u_i,s_j,t_k} is a vector of QoS attribute values acquired from u_i invoking s_j at t_k. If u_i has no experiences on s_j at t_k, $q_{u_i,s_j,t_k} = \emptyset$.

Active user is a user for whom CF algorithm is employed to automatically predict the QoS values of the target Web service.

```
reclassifyingProcess
  for each clusters clu_i
    if the number of elements in clu_i < threshold
        delete the cluster clu_i
        the elements in clu_i will be assigned to
            others closest clusters
```

Fig. 1 Reclassifying process of the enhanced K-means algorithm

3.2 User clusters and service clusters

K-means is a fast in-memory clustering algorithm, and K-means has a time complexity of $O(k^2x)$, where k is the number of clusters and x is the dataset size [24]. K-means begins its clustering by selecting k initial seeds as the temporary cluster centers and then assigning users to the cluster that they are closest to. The centroid of each cluster (which is called representative object) is then taken as the new temporary center and users are reassigned. These steps are repeated until the change in centroid positions fall below a threshold.

In this paper, the similarities between users (services) are used to measure the distances between users (services). Pearson correlation coefficient (PCC) [25] would be employed to calculate the similarities between users (services). The distances between users (services) are increased with the decrease of the similar values between users (services).

Small clusters which contain a few users or services would be build by the K-means algorithm. When clustering CF algorithms are used to predict the QoS values located in the small clusters, the prediction accuracy of clustering CF algorithms would decrease. To address this problem, this paper will delete the small clusters and reclassify the members located in the small clusters. Figure 1 shows the reclassifying process.

We classify the new user into the user cluster which contains most of the users who are located in the same country as the new user, mainly due to the following reasons:

1. According to the experiment results in Tang's paper [23], we can see that every average internal QoS similarity[2] is significantly greater than its corresponding average external QoS similarity.[3]
2. QoS values of Web services are mainly comprised of performance factors such as response time and throughput. Thus, these QoS values are usually highly dependent on the network distance between services and users, i.e., the locations of services and users.

Users' IP would be used to identify the country in which the user is located. Similar to the new user, the new service would be classified into the service cluster which contains most of the services which are located in the same country as the new service.

In order to lower the time complexity of updating clusters, a record will be attached to each user cluster. The record will be used to indicate from which country those users located in this user cluster come and how many users located in this user cluster each country has. In this case, when a new user arrives, the new user can be classified based on the record. Thus, the time complexity of this updating is $O(|U_c|)$, where $|U_c|$ denotes the cardinality of the set U_c. Similar to user cluster, a similar record will also be attached to each service cluster, and the time complexity of new service updating will be $O(|S_c|)$, where $|S_c|$ denotes the cardinality of the set S_c.

3.2.1 Updating clusters

When new users (new services) which will be added to the system arrive or the QoS values for services observed by users have been altered, the users clusters (services clusters) should be updated correspondingly. In order to assure that our approach can be applied to highly dynamic and large-scale environments, the users clusters and services clusters must be updated quickly.

When new user (new service) arrives, the new user (new service) will be assigned to a user cluster (service cluster). Since the user (service) is a new user (service), NULL will be assigned to all items which will be added to the user-service-time tensor. According to the above analysis, the time complexity of new user or new service updating is $O(|U_c|)$ or $O(|S_c|)$.

When the QoS values for services observed by users have been altered, two different kinds of updating approaches would be employed.

If the user (service) isn't representative user (representative service), this user (service) will be reclassified to the cluster that they are closest to. Thus, the distances between this user (service) and the representative user (representative service) of each cluster will be calculated. Thus, the time complexity of this updating is also $O(k)$.

If the user (service) is representative user (representative service), the new representative user (representative service) should be sought from the user cluster (service cluster) where the old representative user (representative service) located in. the time complexity of seeking new representative user (representative service) is $O(|clu_i|^2)$, where clu_i is the cluster where the old representative user (representative service) located, and $|clu_i|$ denotes the number of elements that the cluster clu_i has. Then the similarity between the old representative user (representative service) and representative users (representative services) are calculated, respectively,

[2] For each country, they compute average QoS similarity between every pair of users or services within it, and compute an average value across all countries. This value is called as average internal QoS similarity.

[3] For each country, they also first compute an average QoS similarity between users or services within it and the other users or services outside of the country, and then compute an average value across all countries, which is called as average external QoS similarity.

and then assigning the old representative user (representative service) to the user (service) cluster that it is closest to. After that, the old representative user (representative service) becomes the ordinary user (service). And the time complexity of this process is also $O(k)$. Thus, the time complexity of updating in this case is $O(k + |clu_i|^2)$.

If the probabilities of altering the QoS values for every services (users) are equal, the average time complexity is $O(\frac{(z-k)\times k}{z} + \sum_{i=1}^{k} \frac{(k+|clu_i|^2)}{z})$, where z denotes the total number of users (services). In general, $k \ll z$ and $k \ll |clu_i|$, thus $O(\frac{(z-k)\times k}{z} + \sum_{i=1}^{k} \frac{(k+|clu_i|^2)}{z}) = O(\sum_{i=1}^{k} \frac{|clu_i|^2}{z})$. Since the small clusters have been deleted by our clustering approach, the average time complexity is approximately equal to $O(\overline{|clu|})$, where $\overline{|clu|}$ denotes the average number of elements that a cluster has.

The online updating approach employed by this paper makes sure that the clusters would be updated quickly. But the quality of clusters would be decreased with the increase of the times of execution of online updating process. Thus, the users or services should be reclassified, when the quality of clusters is decreased obviously. And it could be executed off-line.

3.3 Addressing the cold-start problem

The cold-start problem concerns three aspects: (1) new user and old service; (2) new service and old user; (3) new user and new service.

According to the analysis in the fourth paragraph of page 6, CluCF would employ userCluster-service matrix, user-serviceCluster matrix, and userCluster-serviceCluster matrix to address the cold-start problem. Both userCluster-service matrix and user-serviceCluster matrix will be described in Sect. 3.5.3.

3.3.1 New user and old service

Our approach CluCF uses q_{uc_i,s_j} to predict q_{u_i,s_j}, where user u_i is a new user and service s_j isn't a new service, and the entry q_{uc_i,s_j} in userCluster-service matrix $M_{u_c,s}(t_k, d)$ denotes the average QoS values of user cluster uc_i on service s_j. uc_i denotes a user cluster, $u_i \in uc_i$, and $uc_i \in U_c$.

3.3.2 New service and old user

Our approach CluCF uses q_{u_i,sc_j} to predict q_{u_i,s_j}, where service s_j is a new service and user u_i isn't a new user and the entry q_{u_i,sc_j} in user-serviceCluster matrix $M_{u,s_c}(t_k, d)$ denotes the average QoS values of user u_i on service cluster sc_j. sc_j denotes a service cluster, $s_j \in sc_j$, and $sc_j \in S_c$

3.3.3 New user and new service

When service s_j is a new service and user u_i is also a new user, our approach CluCF uses q_{uc_i,sc_j} to predict q_{u_i,s_j}. q_{uc_i,sc_j} belongs to userCluster-serviceCluster matrix $M_{u_c,s_c}(t_k, d)$. Both userCluster-service matrix $M_{u_c,s}(t_k, d)$ and user-serviceCluster matrix $M_{u,s_c}(t_k, d)$ can be converted into userCluster-serviceCluster matrix $M_{u_c,s_c}(t_k, d)$. In this section, we convert userCluster-service matrix $M_{u_c,s}(t_k, d)$ into userCluster-serviceCluster matrix $M_{u_c,s_c}(t_k, d)$. And the formula which is used to get userCluster-serviceCluster matrix $M_{u_c,s_c}(t_k, d)$ is defined as follows:

$$q_{uc_i,sc_j} = \begin{cases} \emptyset, & \text{if nonZero}_{uc_i,sc_j} = \emptyset \\ \dfrac{\sum_{s \in sc_j} q_{uc_i,s}}{\left|nonZero_{uc_i,sc_j}\right|}, & \text{else} \end{cases} \quad (1)$$

where the entry q_{uc_i,sc_j} in userCluster-serviceCluster matrix $M_{u_c,s_c}(t_k, d)$ denotes the average QoS values of user cluster uc_i on service cluster sc_j. And $q_{uc_i,s} \in M_{u_c,s}(t_k, d)$. The entries set $\text{nonZero}_{uc_i,sc_j}$ is defined as follows:

$$\text{nonZero}_{uc_i,sc_j} = \{q_{uc_i,s} \mid s \in sc_j, q_{uc_i,s} \in M_{u_c,s}(t_k, d), \quad q_{uc_i,s} \neq \emptyset\}$$

Thus, when service s_j is a new service and user u_i is also a new user, our approach CluCF uses q_{uc_i,sc_j} to predict q_{u_i,s_j}.

3.4 Overview of the algorithm

In this section, an abstract description of CluCF has been given. The CluCF algorithm comprises the following six sub-procedures.

1. Convert the user-service-time tensor $M_{u,s,t}$ into a user-service matrix $M_{u,s}(t_k, d)$. This converting would be done off-line.
2. The user-service matrix $M_{u,s}(t_k, d)$ is used to classify users (services). The users clusters (services clusters) will be build in this procedure. And it would be done off-line.
3. Convert the user-service matrix $M_{u,s}(t_k, d)$ into userCluster-service matrix $M_{u_c,s}(t_k, d)$ and user-serviceCluster matrix $M_{u,s_c}(t_k, d)$, respectively. These converting would be done off-line.
4. Calculate the similarity between target service and all other services located in the same cluster with target service, respectively. And then select the top K services with highest similarity to the target service.
5. Calculate the user similarity between active user and all other users located in the same cluster with active user, respectively. And then select the top K users with highest similarity to the active user.

6. Combine user-based and item-based methods to predict the missing values (QoS values) for the active user.

3.5 Matrix converting

In this section, we present the process of converting the user-service-time tensor $M_{u,s,t}$ into a user-service matrix $M_{u,s}(t_k, d)$ and then convert the user-service matrix $M_{u,s}(t_k, d)$ into userCluster-service matrix $M_{u_c,s}(t_k, d)$ and user-serviceCluster matrix $M_{u,s_c}(t_k, d)$, respectively.

3.5.1 Converting user-service-time tensor into user-service matrix

In a typical CF scenario, a general Web services recommender system consists of N users and M services, and the relationship between users and services is denoted by an $N \times M$ matrix, called the user-service matrix. Every entry $q_{n,m}$ in this matrix represents a vector of QoS values (e.g., response time, failure rate), which is observed by the user n on the service m.

But we found that the QoS values which are observed by the same user on the same service at different time would be different. Thus, in this paper, in order to improve the QoS prediction accuracy, user-service-time tensor $M_{u,s,t}$ is employed to record the historical QoS values, which is observed by the user on the service at a certain time.

In fact, the number of services far exceeds what any user can hope to absorb, and thus, user-service-time tensor $M_{u,s,t}$ is very sparse. In order to increase the density of data of matrix, we convert user-service-time tensor $M_{u,s,t}$ into a user-service matrix $M_{u,s}(t_k, d)$. And the formula which is used to get user-service matrix $M_{u,s}(t_k, d)$ is defined as follows:

$$q_{u_i,s_j}(t_k, d) = \begin{cases} 0, & \text{if } \text{nonZero}_{u_i,s_j}(t_k, d) = \emptyset \\ \dfrac{\sum_{t \in T_{t_k,d}} q_{u_i,s_j,t}}{\left|\text{nonZero}_{u_i,s_j}(t_k,d)\right|}, & else \end{cases} \tag{2}$$

where the entry $q_{u_i,s_j}(t_k, d)$ in user-service matrix $M_{u,s}(t_k, d)$ denotes the average QoS values. Both t_k and d are parameters for user-service matrix $M_{u,s}(t_k, d)$. t_k is determined by the missing QoS values which will be predicted by our approach. For example, if the entry which will be predicted is q_{u_1,s_1,t_x}, t_k equals to t_x.

A tunable parameter d is used to adjust the prediction accuracy. On the one hand, the density of matrix $M_{u,s}(t_k, d)$ tends to increase with the increase of parameter d. It tends to improve the prediction accuracy. On the other hand, the absolute value of $q_{u_i,s_j,t_k} - q_{u_i,s_j}(t_k, d)$ tends to increase with the increase of parameter d. It tends to decrease the prediction accuracy. Thus, a proper parameter value for d is important for our approach. The entry $q_{u_i,s_j,t}$ in user-service-time tensor $M_{u,s,t}$ denotes the real QoS value, which is observed by the user u_i on the service s_j at time t. Time interval set $T_{t_k,d}$ is defined as follows:

$$T_{t_k,d} = \begin{cases} \{t_{k'} | k - d \le k' \le k + d, 1 \le k - d, k + d \le r\} \\ \{t_{k'} | k - d \le k' \le r, 1 \le k - d, k + d > r\} \\ \{t_{k'} | 1 \le k' \le k + d, 1 > k - d, k + d \le r\} \\ \{t_{k'} | 1 \le k' \le r, 1 > k - d, k + d > r\} \end{cases} \tag{3}$$

Time interval set $T_{t_k,d}$ is determined by parameter t_k and d. For example, when d = 2 and $t_k = t_2$, we can get $T_{t_2,2} = \{t_1, t_2, t_3, t_4\}$, according to formula 3. The entries set $\text{nonZero}_{u_i,s_j}(t_k, d)$ is defined as follows:

$$\begin{aligned} &nonZero_{u_i,s_j}(t_k,d) \\ &= \{q_{u_i,s_j,t} | q_{u_i,s_j,t} \in M_{u,s,t},\ t \in T_{t_k,d},\quad q_{u_i,s_j,t} \neq 0\} \end{aligned} \tag{4}$$

In order to increase the density of data of matrix, the user-service-time tensor $M_{u,s,t}$ can be converted into a user-service matrix $M_{u,s}(t_k, d)$ by formulas 2, 3, and 4.

3.5.2 Updating user-service matrix

When new users (new services) which will be added to the system arrive or the QoS values for services observed by users at time intervals have been altered, the user-service matrix should be updated correspondingly.

When a new user arrives, there are $1 \times m \times r$ items which will be added to the user-service-time tensor. m refers to the total number of Web services collected by the recommendation system, and r refers to the total number of time intervals. Since the user is a new user, zero will be assigned to all items which will be added to the user-service-time tensor. Thus, for the new user, zero will also be assigned to all items which will be added to the user-service matrix. Thus, in this case, the time complexity of updating user-service matrix is $O(1)$.

When a new service arrives, the process of updating user-service matrix is similar to the process of updating for new user. Thus, the time complexity of updating user-service matrix for new service is also $O(1)$.

Thus, when a new user or service arrives, the time complexity of updating user-service matrix is $O(1)$.

When the QoS value for a service observed by a user at a time interval has been altered, only one item will be updated in user-service matrix. According to formula 2, the time complexity of updating is $O(d)$ in this case. d is a parameter for user-service matrix $M_{u,s}(t_k, d)$.

3.5.3 Converting user-service matrix into userCluster-service matrix and user-serviceCluster matrix, respectively

Sometimes the density of data of matrix isn't enough just by converting user-service-time tensor $M_{u,s,t}$ into user-service matrix $M_{u,s}(t_k, d)$. Thus, in order to increase the density further, we need to convert user-service matrix $M_{u,s}(t_k, d)$ into userCluster-service matrix $M_{u_c,s}(t_k, d)$ and user-serviceCluster matrix $M_{u,s_c}(t_k, d)$, respectively. And Fig. 2 shows the process of converting user-service matrix into userCluster-service matrix and user-serviceCluster matrix.

The formula which is used to get userCluster-service matrix $M_{u_c,s}(t_k, d)$ is defined as follows:

$$q_{uc_i,s_j}(t_k,d) = \begin{cases} 0, & \text{if } \text{nonZero}_{uc_i,s_j}(t_k,d) = \emptyset \\ \dfrac{\sum_{u \in uc_i} q_{u,s_j}(t_k,d)}{\left|\text{nonZero}_{uc_i,s_j}(t_k,d)\right|}, & \text{else} \end{cases} \tag{5}$$

where the entry $q_{uc_i,s_j}(t_k, d)$ in userCluster-service matrix $M_{u_c,s}(t_k, d)$ denotes the average QoS values of user cluster uc_i on service s_j. uc_i denotes a user cluster and $uc_i \in U_c$. And $q_{u,s_j}(t_k, d) \in M_{u,s}(t_k, d)$. The entries set $\text{nonZero}_{uc_i,s_j}(t_k, d)$ is defined as follows:

$$\begin{aligned} \text{nonZero}_{uc_i,s_j}(t_k,d) = \{ & q_{u,s_j}(t_k,d) | \\ & q_{u,s_j}(t_k,d) \in M_{u,s}(t_k,d), \quad u \in uc_i, \quad q_{u,s_j}(t_k,d) \neq 0\} \end{aligned} \tag{6}$$

The formula which is used to get user-serviceCluster matrix $M_{u,s_c}(t_k, d)$ is defined as follows:

$$q_{u_i,sc_j}(t_k,d) = \begin{cases} 0, & \text{if } \text{nonZero}_{u_i,sc_j}(t_k,d) = \emptyset \\ \dfrac{\sum_{s \in sc_j} q_{u_i,s}(t_k,d)}{\left|\text{nonZero}_{u_i,sc_j}(t_k,d)\right|}, & else \end{cases} \tag{7}$$

where the entry $q_{u_i,sc_j}(t_k, d)$ in user-serviceCluster matrix $M_{u,s_c}(t_k, d)$ denotes the average QoS values of user u_i on service cluster sc_j. sc_j denotes a service cluster and $sc_j \in S_c$. And $q_{u_i,s}(t_k, d) \in M_{u,s}(t_k, d)$. The entries set $\text{nonZero}_{u_i,sc_j}(t_k, d)$ is defined as follows:

$$\begin{aligned} \text{nonZero}_{u_i,sc_j}(t_k,d) = \{ & q_{u_i,s}(t_k,d) | \\ & q_{u_i,s}(t_k,d) \in M_{u,s}(t_k,d), \ s \in sc_j, \ q_{u_i,s}(t_k,d) \neq 0\} \end{aligned} \tag{8}$$

The user-service matrix $M_{u,s}(t_k, d)$ can be converted into the userCluster-service matrix $M_{u_c,s}(t_k, d)$ and user-serviceCluster matrix $M_{u,s_c}(t_k, d)$ to increase the density further by formulas 5, 6, 7, and 8.

3.5.4 Updating userCluster-service matrix and user-serviceCluster matrix

When new users (new services) which will be added to the system arrive or the QoS values for services observed by users at time intervals have been altered, the userCluster-service matrix and user-serviceCluster matrix should be updated correspondingly.

When a new user u_{i_1} arrives, userCluster-service matrix and user-serviceCluster matrix would be updated correspondingly. According to the analysis in Sects. 3.2.1 and

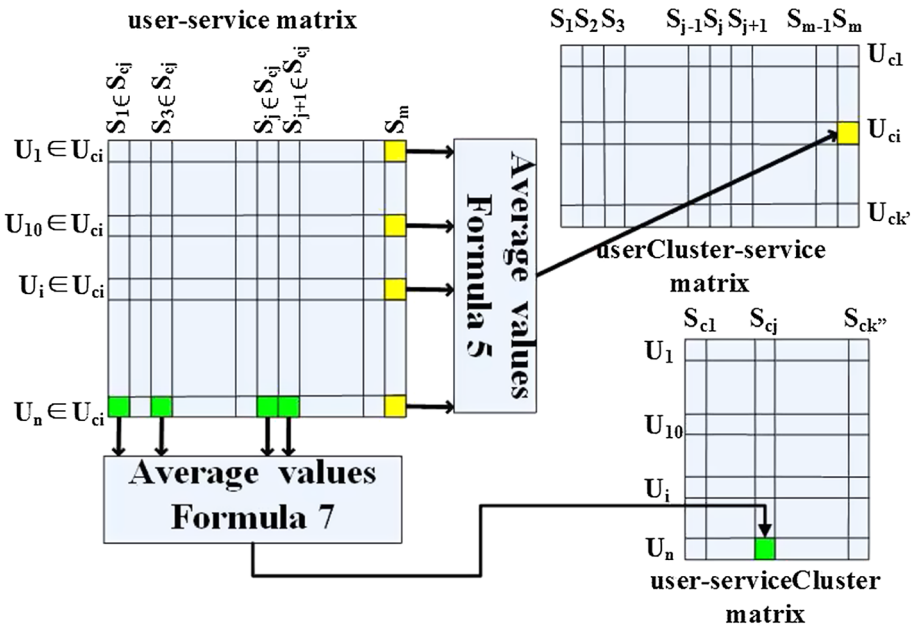

Fig. 2 Process of converting user-service matrix into userCluster-service matrix and user-serviceCluster matrix

3.5.2, for the new user, zero will be assigned to all items which will be added to the user-service-time tensor and user-service matrix, and the new user will be classified into the user cluster which contains most of the users who are located in the same country as the new user. Thus, all items in userCluster-service matrix will not be altered. And zero will be assigned to all items which will be added to the user-serviceCluster matrix. Thus, the time complexity of updating userCluster-service matrix is $O(0)$ and the time complexity of updating user-serviceCluster matrix is $O(1)$.

When a new service arrives, the process of updating user-service matrix is similar to the process of updating for new user. Thus, the time complexity of updating userCluster-service matrix is $O(1)$ and the time complexity of updating user-serviceCluster matrix is $O(0)$.

When the QoS value for a service observed by a user at a time interval has been altered, an item in userCluster-service matrix and an item in user-serviceCluster matrix should be updated correspondingly. According to Sect. 3.5.3, the time complexity of updating userCluster-service matrix is $O(\sum_{i=1}^{k'} \frac{|uc_i|^2}{n})$, where k' is the number of users clusters that users have been classified, n refers to the total number of service users registered in the recommendation system, and $|uc_i|$ denotes the number of users that user cluster uc_i has. And time complexity of updating user-serviceCluster matrix is $O(\sum_{j=1}^{k''} \frac{|sc_j|^2}{m})$, where k'' is the number of services clusters that services have been classified, m refers to the total number of services registered in the recommendation system, and $|sc_j|$ denotes the number of services that service cluster sc_j has.

Since the small clusters have been deleted by our clustering approach, the average time complexity of updating userCluster-service matrix is approximately equal to $O(\overline{|uc|})$, where $\overline{|uc|}$ denotes the average number of elements that a user cluster has. And the average time complexity of updating user-serviceCluster matrix is approximately equal to $O(\overline{|sc|})$, where $\overline{|sc|}$ denotes the average number of elements that a service cluster has.

3.6 Similarity measure

Similarity computation between services or users is a critical step in memory-based collaborative filtering algorithms. There are a number of different methods to compute the similarity between services or users. For example, cosine-based similarity [25], Pearson correlation coefficient (PCC) [25], and adjusted-cosine similarity [25].

The Pearson correlation coefficient-based CF algorithm is a representative CF algorithm and is widely used in the CF research community [9]. Although many variations of Pearson correlation coefficient have been proposed, such as constrained Pearson correlation [9], Spearman rank correlation [9], and Kendall's τ correlation [9], Pearson correlation coefficient is constantly employed by a large number of CF algorithms until now, such as [23,26]. Thus, in this paper, Pearson correlation coefficient was employed to calculate the similarity between services or users.

PCC often overestimates the similarities of services which are actually not similar but happen to have similar QoS on a few common users [27]. To address this problem, this paper employs a similarity weight to reduce the influence of the small number of similar common users. An enhanced PCC for $sim(s_{j_1}, s_{j_2})$ is defined as follows:

$$\begin{aligned} \mathrm{sim}(s_{j_1}, s_{j_2}) = {} & \frac{\left|U_c^{s_{j_1}, s_{j_2}}\right|}{|U_c|} \\ & \times \frac{\sum_{uc \in U_c^{s_{j_1}, s_{j_2}}} (q_{uc,s_{j_1}} - \overline{q_{s_{j_1}}})(q_{uc,s_{j_2}} - \overline{q_{s_{j_2}}})}{\sqrt{\sum_{uc \in U_c^{s_{j_1}, s_{j_2}}} (q_{uc,s_{j_1}} - \overline{q_{s_{j_1}}})^2} \sqrt{\sum_{uc \in U_c^{s_{j_1}, s_{j_2}}} (q_{uc,s_{j_2}} - \overline{q_{s_{j_2}}})^2}} \end{aligned} \tag{9}$$

The entry in userCluster-service matrix $M_{u_c,s}(t_k, d)$ will be used to calculate the similarity between services, where $U_c^{s_{j_1}, s_{j_2}}$ is the subset of user clusters U_c. And $U_c^{s_{j_1}, s_{j_2}} = \{uc | uc \in U_c, q_{uc,s_{j_1}} \neq 0, q_{uc,s_{j_2}} \neq 0\}$. And $\frac{\left|U_c^{s_{j_1}, s_{j_2}}\right|}{|U_c|}$ is a similarity weight. When $U_c^{s_{j_1}, s_{j_2}}$ is small, the similarity weight will devalue the similarity estimation between the services. Both $q_{uc,s_{j_1}}$ and $q_{uc,s_{j_2}}$ are the entry in userCluster-service matrix $M_{u_c,s}(t_k, d)$. $\overline{q_{s_{j_1}}}$ represents the vector of average QoS values of the Web service s_{j_1} in userCluster-service matrix $M_{u_c,s}(t_k, d)$. $sim(s_{j_1}, s_{j_2})$ ranges from $[-1,1]$ with a larger value indicating that services s_{j_1} and s_{j_2} are more similar.

Similar to calculating the similarity between services, an enhanced PCC is also employed to compute the similarity between users. The similarity computation of two users u_{i_1} and u_{i_2} can be described as:

$$\begin{aligned} \mathrm{sim}(u_{i_1}, u_{i_2}) = {} & \frac{\left|S_c^{u_{i_1}, u_{i_2}}\right|}{|S_c|} \\ & \times \frac{\sum_{sc \in S_c^{u_{i_1}, u_{i_2}}} (q_{u_{i_1},sc} - \overline{q_{u_{i_1}}})(q_{u_{i_2},sc} - \overline{q_{u_{i_2}}})}{\sqrt{\sum_{sc \in S_c^{u_{i_1}, u_{i_2}}} (q_{u_{i_1},sc} - \overline{q_{u_{i_1}}})^2} \sqrt{\sum_{sc \in S_c^{u_{i_1}, u_{i_2}}} (q_{u_{i_2},sc} - \overline{q_{u_{i_2}}})^2}} \end{aligned} \tag{10}$$

The entry in user-serviceCluster matrix $M_{u,s_c}(t_k, d)$ will be used to calculate the similarity between users, where $S_c^{u_{i_1}, u_{i_2}}$ is the subset of service clusters S_c. And $S_c^{u_{i_1}, u_{i_2}} = \{sc | sc \in S_c, q_{u_{i_1},sc} \neq 0, q_{u_{i_2},sc} \neq 0\}$. And $\frac{\left|S_c^{u_{i_1}, u_{i_2}}\right|}{|S_c|}$ is a similarity weight. Both $q_{u_{i_1},s_c}$ and $q_{u_{i_2},s_c}$ are the entry in user-serviceCluster matrix $M_{u,s_c}(t_k, d)$. $\overline{q_{u_{i_1}}}$ represents the vector of average QoS values observed by the user u_{i_1} in user-serviceCluster matrix $M_{u,s_c}(t_k, d)$. $sim(u_{i_1}, u_{i_2})$ also ranges

from [−1,1] with a larger value indicating that users u_{i_1} and u_{i_2} are more similar.

3.7 Similar neighbors selection

After computing the similarities between active user and all other users who are located in the same user cluster as active user, a user similarity vector has been obtained. Likewise, a service similarity vector has also been obtained.

Thus, the TopK similar users (services) would be selected from the user (service) similarity vector to construct a set of the TopK similar users (services) to active user (target service) by traditional Top-K algorithm. $N(u)$ is used to denote the set of the TopK similar users to active user u. $N(s)$ is used to denote the set of the TopK similar services to target service s.

Similar to [19], the similar neighbors whose similarity is equal to or smaller than 0 will be removed. Thus, a new set of the TopK similar users $N'(u)$to active user u would be obtained by removing those similar neighbors whose similarity is equal to or smaller than 0 from the set of the TopK similar users $N(u)$. And a new set of the TopK similar services $N'(s)$ to target service s would also be obtained by similar method.

3.8 Missing value prediction

The following formula is employed to predict the missing QoS values for the active user by user-based CF algorithms:

$$p_u(u,s,t) = \overline{q_u} + \frac{\sum_{u_i \in N'(u)} \mathrm{sim}(u,u_i)(q_{u_i,s} - \overline{q_{u_i}})}{\sum_{u_i \in N'(u)} \mathrm{sim}(u,u_i)} \tag{11}$$

The entry in user-service matrix $M_{u,s}(t,d)$ will be used to calculate the missing QoS values $p_u(u,s,t)$ by formula 11, where $\overline{q_u}$ represents the vector of average QoS values observed by the user u in user-service matrix $M_{u,s}(t,d)$. And $q_{u_i,s}$ represents the vector of QoS values observed by the user u_i on service s in user-service matrix $M_{u,s}(t,d)$.

The following formula is employed to predict the missing QoS values by item-based CF algorithms:

$$p_s(u,s,t) = \overline{q_s} + \frac{\sum_{s_j \in N'(s)} \mathrm{sim}(s,s_j)(q_{u,s_j} - \overline{q_{s_j}})}{\sum_{s_j \in N'(s)} \mathrm{sim}(s,s_j)} \tag{12}$$

The entry in user-service matrix $M_{u,s}(t,d)$ will be used to calculate the missing QoS values $p_s(u,s,t)$ by formula 12, where $\overline{q_s}$ represents the vector of average QoS values of Web service s in user-service matrix $M_{u,s}(t,d)$. And q_{u,s_j} represents the vector of QoS values observed by the user u on service s_j in user-service matrix $M_{u,s}(t,d)$.

In order to improve the prediction accuracy, a hybrid CF algorithm is employed, which systematically combines user-based and item-based methods to predict the missing QoS values. Since these two predicted values may have different prediction performance, two influence weights, inf_u and inf_s, are employed to balance these two predicted values. We combine the two methods by using formula 13:

$$p(u,s,t) = \inf_u \times p_u(u,s,t) + \inf_s \times p_s(u,s,t) \tag{13}$$

where $inf_u + inf_s = 1$.

inf_u is defined as follows:

$$inf_u = \frac{\sum_{u_i \in N'(u)} \mathrm{sim}(u,u_i)}{\sum_{u_i \in N'(u)} \mathrm{sim}(u,u_i) + \sum_{s_j \in N'(s)} sim(s,s_j)} \tag{14}$$

A higher value indicates a higher prediction accuracy of $p_u(u,s,t)$.

inf_s is defined as follows:

$$inf_s = \frac{\sum_{s_j \in N'(s)} \mathrm{sim}(s,s_j)}{\sum_{u_i \in N'(u)} sim(u,u_i) + \sum_{s_j \in N'(s)} \mathrm{sim}(s,s_j)} \tag{15}$$

A higher value indicates a higher prediction accuracy of $p_s(u,s,t)$.

Influence weights, inf_u and inf_s, determine how much the hybrid method relies on the user-based prediction and the item-based prediction automatically. Our approach are the first one to employ influence weights to balance $p_u(u,s,t)$ and $p_s(u,s,t)$, automatically. And the experimental results in Sect. 4 show that they would significantly improve the QoS value prediction accuracy.

3.9 Complexity analysis

The main off-line computation of our approach is converting user-service-time tensor into user-service matrix and user-service matrix into userCluster-service matrix and user-serviceCluster matrix. And the main on-line computation of our approach is constructing the set of the TopK similar users $N'(u)$ and the set of the TopK similar services $N'(s)$.

Based on the formula 2, the time complexity of converting user-service-time tensor into user-service matrix is $O(n \times m \times r)$ and the time complexity of updating the user-service matrix is $O(r)$, where n refers to the total number of users registered in the recommendation system, m refers to the total number of services registered in the recommendation system, and r refers to the total number of time interval.

Based on the formula 5, the time complexity of converting user-service matrix into userCluster-service matrix is $O(n \times m)$. Based on the formula 7, the time complexity of converting user-service matrix into user-serviceCluster matrix is $O(n \times m)$.

Thus, the time complexity of converting matrices is $O(n \times m \times r + n \times m + n \times m) = O(n \times m \times r)$. And these works would be done off-line.

Based on the formula 9 and traditional Top-K algorithm, the average time complexity of constructing the set of the TopK similar services $N'(s)$ is $O(|U_c| \times \overline{|sc|})$. Based on the formula 10 and traditional Top-K algorithm, the average time complexity of constructing the set of the TopK similar users $N'(u)$ is also $O(\overline{|uc|} \times |S_c|)$. Thus, the average time complexity of predicting QoS values is $O((|U_c| \times \overline{|sc|}) + (\overline{|uc|} \times |S_c|))$, where $\overline{|uc|} \ll n$, $|U_c| \ll n$, $\overline{|sc|} \ll m$ and $|S_c| \ll m$.

The above complexity analysis shows that our approach is very efficient and can be applied to large-scale systems.

According to the analysis in Sects. 3.2.1, 3.5.2, and 3.5.4, when new user (new service) arrives, the time complexity of updating models is $O(|U_c|)$ or $O(|S_c|)$. And when the QoS value for a service observed by a user at a time interval has been altered, the time complexity of updating models is $O(d + \overline{|uc|} + \overline{|sc|})$. $\overline{|uc|}$ denotes the average number of elements that a user cluster has. $\overline{|sc|}$ denotes the average number of elements that a service cluster has. d is a parameter for user-service matrix $M_{u,s}(t_k, d)$. Generally, $d \ll \overline{|uc|}$ *and* $d \ll \overline{|sc|}$, thus, when the QoS value for a service observed by a user at a time interval has been altered, the time complexity of updating models is $O(\overline{|uc|} + \overline{|sc|})$.

Thus, the major disadvantage of model-based CF algorithms, namely time-consuming to update, could be avoided by employing our approach, according to the above analysis. This property assures that our approach can be applied to those environments which are highly dynamic and large-scaled, such as Web service recommendation systems.

In short, our approach can be applied to highly dynamic and large-scale environments.

4 Experiments

All experiments were implemented and deployed with JDK6.0 and Eclipse Helios Service Release 2.

All experiments were run on a win7-based PC with Intel Core i3-3220 CPU having a speed of 3.3 GHz and 4 GB of RAM.

Two experiments have been implemented. The purpose of the first experiment is to analyze the effect of our approach to alleviate the data sparsity problem. The purpose of the second experiment is to analyze the impact of parameter d on prediction accuracy.

4.1 Dataset

To evaluate our approach in the real world, this paper adopts a real-world Web service dataset WSDream-dataset 3,[4] which was published in references [3].

The original dataset 3 contains QoS records of service invocations on 4532 Web services from 142 users at 64 time interval, which are transformed into a user-service-time tensor. Every time interval lasts for 15 min. Then the dataset is a $142 \times 4532 \times 64$ user-service-time tensor, and each item is a pair values (RT, TP). RT denotes response time, and TP denotes throughput. The original user-service-time tensor will be decomposed into two simpler matrices: $142 \times 4532 \times 64$ user-service-time RT tensor and $142 \times 4532 \times 64$ user-service-time TP tensor.

In the following experiment, the 142 users are divided into two groups: 118 service users randomly selected as training service users and the rest as test service users. The RT matrix is divided into two parts: the RT-training matrix and the RT-test matrix, and so is the TP matrix. Each experiment is performed 100 times, and their average values are taken as results.

4.2 Evaluation metrics

Statistical accuracy metrics evaluate the accuracy of a system by comparing the numerical prediction QoS values with the actual QoS values acquired by invoking actual service. Mean absolute error (MAE) [28] between actuality and predictions is a widely used metric. MAE is a measure of the deviation of predictions from their actual QoS values. For each actuality-prediction pair $< Aq_{u_i,s_j,t_k}, Pq_{u_i,s_j,t_k} >$, this metric treats the absolute error between them equally. The MAE is computed by first summing these absolute errors of the N corresponding actuality-prediction pairs and then computing the average. Formally,

$$MAE = \frac{\sum_{u_i,s_j,t_k} \left| Aq_{u_i,s_j,t_k} - Pq_{u_i,s_j,t_k} \right|}{N}$$

where Aq_{u_i,s_j,t_k} denotes actual QoS values of Web service s_j observed by user u_i at time interval t_k and Pq_{u_i,s_j,t_k} denotes the predicted QoS values of Web service s_j observed by user u_i at time interval t_k, and N denotes the number of predicted values. The lower the MAE, the higher the recommendation system accuracy.

Since different QoS properties of Web services have distinct value ranges, the normalized mean absolute error (NMAE) [28] metric is also used to measure the prediction accuracy. NMAE is defined as:

[4] http://www.wsdream.net/wsdream/dataset.html.

$$NMAE = \frac{MAE}{(\sum_{u_i,s_j,t_k} Aq_{u_i,s_j,t_k})/N}$$

Lower NMAE value represents higher prediction accuracy about recommendation system.

In these papers, NMAE is employed to measure the prediction quality, since it is most commonly used and easiest to interpret directly.

4.3 Impact of tensor (matrix) density

In order to show the effectiveness of our method to alleviate the data sparsity problem and justify the usage of the proposed CluCF algorithm, we compare the prediction accuracy of our method with some other famous methods under different tensor density. Those methods are user-based CF [29], item-based CF [30], WSRec [19], and WSPred [3].

User-based CF and item-based CF are famous memory-based CF algorithms. Since user-based CF and item-based CF have been widely accepted and adopted, they have been picked up for comparison in a number of CF-based approaches. Thus, they are also picked up for comparison in this paper. WSRec is a well-known hybrid algorithm combing user-based CF and item-based CF, and the prediction accuracy of WSRec is quite high. Similar to WSRec, our approach is also a hybrid algorithm combing user-based CF and item-based CF. Thus, WSRec is picked up for comparison in this paper. In order to show the improvement of addressing the data sparsity problem, our approach should be compared with a famous model-based CF algorithm which has relatively high performance to address the data sparsity problem. Since WSPred is such an algorithm, it is picked up for comparison in this paper.

We change the tensor density from 5 to 40 % with a step value of 5 %. The tensors with missing values are in different densities. For example, 5 % means that we randomly remove 95 % entries from the original tensor and use the remaining 5 % entries to predict.

The parameters to our approach CluCF are $topK = 30$, $d = 9$, $k' = 3$, $k'' = 300$. $thresholdUser = 20$ and $thresholdServie = 80$. *thresholdUser* and *thresholdService* are the variable *threshold* which is used in Fig. 1.

The parameters to WSPred are $l = 15$ and $\lambda_1 = \lambda_2 = \lambda_3 = \eta = 0.001$.

Since user-based CF, item-based CF, and WSRec predict QoS values by employing user-service matrix, the original user-service-time tensor must be converted into user-service matrix. In order to simulate the real-world situation, a entry $q_{u_{i_1},s_{j_1}}$ of user-service matrix, equals the entry $q_{u_{i_1},s_{j_1},t_{k_1}}$ of user-service-time tensor, where t_{k_1} will be randomly selected from set $\{1, 2, \ldots, 64\}$. And the parameter to them are $topK = 30$. We change the matrix density from 5 to 40 % with a step value of 5 %. The matrices with missing values are in different densities. For example, 5 % means that we randomly remove 95 % entries from the original matrix and use the remaining 5 % entries to predict. And the parameter to WSRec is $\lambda = 0.7$.

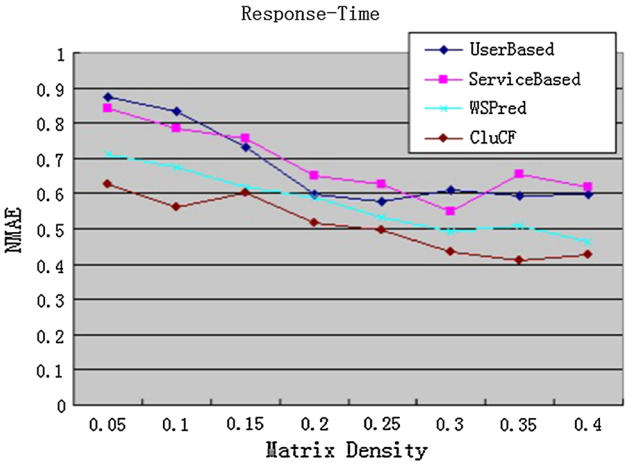

Fig. 3 Impact of tensor density

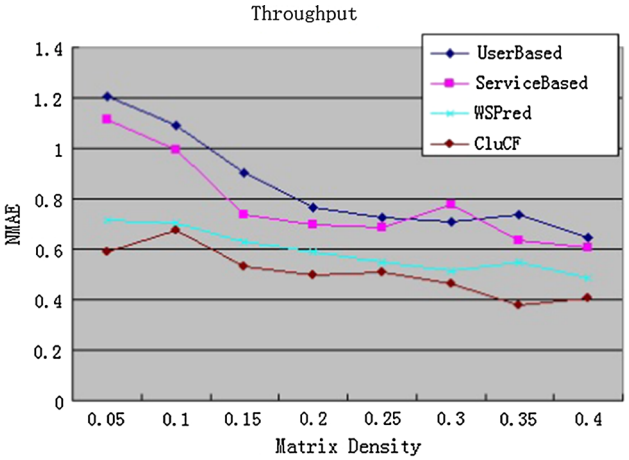

Fig. 4 Impact of tensor density

Figures 3 and 4 show the experimental results of response time and throughput, respectively. The experimental results show that our approach CluCF achieves higher prediction accuracy (lower NMAE value) than other competing methods under different tensor (matrix) density.

In the experimental results, we observe that the performance of user-based CF, item-based CF, and WSRec is worse than that of other methods. The reason is that if the matrix density is sparse, similar users may have too few ratings in common or may even show a negative correlation due to a small number of ratings in common.

Our approach CluCF improves matrix density by converting the user-service-time tensor into a user-service matrix and then converting the user-service matrix into userCluster-service matrix and user-serviceCluster matrix.

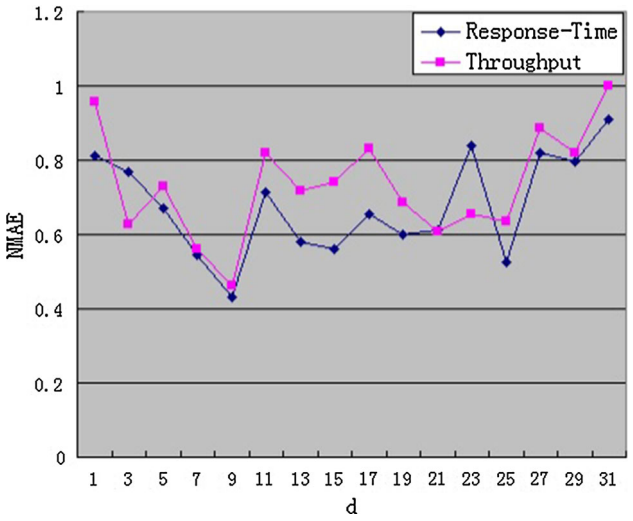

Fig. 5 Impact of parameter d

Then our approach CluCF predicts QoS values by employing userCluster-service matrix and user-serviceCluster matrix. The densities of userCluster-service matrix and user-service Cluster matrix are much higher than user-service-time tensor. Moreover, our approach CluCF systematically combines user-based and item-based methods to predict the missing QoS values and employs influence weights, inf_u and inf_s, to balance these two predicted values, automatically. Thus, our approach CluCF achieves higher prediction accuracy (lower NMAE value) than other approaches under different tensor (matrix) density.

4.4 Impact of parameter d

To study the impact of parameter d, two experiments have been implemented. The parameters are $topK = 30$, $k' = 3$, $k'' = 300$. $thresholdUser = 20$, and $thresholdUser = 80$. And the tensor density is 30 %. Figure 5 shows the experimental results. We observe that in Fig. 5, as d increases, the NMAE decreases, but when d surpasses a certain threshold, the NMAE increases with further increase of d. The reason is that, on the one hand, the matrix density tends to increase with the increase of parameter d. It tends to improve the prediction accuracy. On the other hand, the absolute value of $q_{u_i,s_j,t_k} - q_{u_i,s_j}(t_k, d)$ tends to increase with the increase of parameter d. It tends to decrease the prediction accuracy. When the value of d is at a high level, the increase of matrix density tends to slow down even if the value of the d is still raising, while the increase of absolute value of $q_{u_i,s_j,t_k} - q_{u_i,s_j}(t_k, d)$ may remain at the same level or tend to be augmented. Thus, a proper parameter value for d is important.

5 Conclusions

Collaborative filtering is an effective method for Web service selection and recommendation. But they suffer from two fundamental problems: sparsity and scalability.

Model-based CF algorithms could address the data sparsity problem. But those approaches have a severe drawback. The time complexity of updating for those approaches are much higher. For example, the time complexity of SVD-updating is $O(f^2M + f^2N)$. Thus, those approaches aren't fit for highly dynamic and large-scale environments, such as Web service recommendation systems. In order to overcome this drawback, the CluCF has been proposed in this paper. CluCF improves matrix density by converting the user-service-time tensor into a user-service matrix and then converting the user-service matrix into userCluster-service matrix and userserviceCluster matrix to alleviate the data sparsity problem. According to the analysis in Sect. 3.9, the time complexity of updating for CluCF is $O(\overline{|uc|} + \overline{|sc|})$ at most. It assures that our approach CluCF can be applied to those environments which are highly dynamic and large-scaled, such as Web service recommendation systems. The experiment results in Sect. 4.3 show that CluCF is an effective approach to alleviate the data sparsity problem.

Our approach CluCF is a clustering CF algorithm. Clustering CF algorithms would address the scalability problem, but there are trade-offs between scalability and prediction performance. In order to improve the prediction accuracy, our approach employs time factor. Moreover, our approach systematically combines user-based and item-based methods and employs influence weights to balance these two predicted values, automatically.

In this paper, K-means is employed to classify users and Web services; meanwhile, PCC is employed to measure the distance between users or Web services. Although other clustering algorithms would be combined with the approach proposed by this paper in order to alleviate the data sparsity problem, the actual effect still needs further testing. Thus, more experiments should be done. However, it is beyond the scope of this paper. And it will be discussed in future work. In this paper, we focus on the effect of K-means algorithm composed with the approach to improve the matrix density.

In future work, our approach will be improved to alleviate gray sheep and shilling attack problem.

Acknowledgements The work described in this paper was supported by the National Natural Science Foundation of China under Grant Nos. 91118004, 61232007 and the Innovation Program of Shanghai Municipal Education Commission (No. 13ZZ023).

References

1. Yu T, Zhang Y, Lin K-J (2007) Efficient algorithms for web services selection with end-to-end qos constraints. ACM Trans Web. doi:10.1145/1232722.1232728
2. Zhang LJ, Zhang J, Cai H (2007) Services computing. Springer and Tsinghua University Press, New York
3. Zhang Yilei, Zheng Zibin, Lyu Michael R (2011) WSPred: a time-aware personalized QoS prediction framework for web services. In: Proceedings of IEEE symposium on software reliability engineering. doi:10.1109/ISSRE.2011.17
4. Shardanand U, Maes P (1995) Social information filtering: algorithms for automating 'Word of Mouth'. CHI '95 Proceedings of the SIGCHI conference on human factors in computing systems. doi:10.1145/223904.223931
5. Hill W, Stead L, Rosenstein M, Furnas G (1995) Recommending and evaluating choices in a virtual community of use. CHI '95 Proceedings of the SIGCHI conference on human factors in computing systems. doi:10.1145/223904.223929
6. Konstan J, Miller B, Maltz D, Herlocker J, Gordon L, Riedl J (1997) GroupLens: applying collaborative filtering to usenet news. Commun ACM 40:77–87. doi:10.1145/245108.245126
7. Rich E (1979) User modeling via stereotypes. Cogn Sci 3:329–354
8. Linden G, Smith B, York J (2003) Amazon.com recommendations: item-to-item collaborative filtering. IEEE Internet Comput. doi:10.1109/MIC.2003.1167344
9. Su X, Khoshgoftaar TM (2009) A survey of collaborative filtering techniques. Adv Artif Intell. doi:10.1155/2009/421425
10. Wu J, Chen L, Xie Y et al. Titan: a system for effective web service discovery[C]. In: Proceedings of the 21st international conference companion on World Wide Web. ACM, 2012, pp 441–444
11. Breese JS, Heckerman D, Kadie C (1998) Empirical analysis of predictive algorithms for collaborative filtering. In: UAI
12. Billsus D, Pazzani M (1998) Learning collaborative information filters. In: Proceedings of the 15th international conference on machine learning (ICML98)
13. Sarwar et al. (2002) Incremental singular value decomposition algorithms for highly scalable recommender systems. In: Fifth international conference on computer and information science
14. Berry M et al (1995) Using linear algebra for intelligent information retrieval. SIAM Rev 37(4):573–595
15. Sarwar BM, Karypis G, Konstan JA, Riedl J (2000) Application of dimensionality reduction in recommender system—a case study. In: ACM WebKDD workshop
16. Sarwar B M, Konstan J A, Borchers A et al (1998) Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In: Proceedings of the ACM conference on computer supported cooperative work. ACM 1998:345–354
17. Balabanovic M, Shoham Y (1997) Fab: Content-based collaborative recommendation. Commun ACM 40(3):66–72
18. Claypool M, Gokhale A, Miranda T, et al., (1999) Combining content-based and collaborative filters in an online newspaper. In: Proceedings of the SIGIR workshop on recommender systems: algorithms and evaluation, Berkeley, CA, USA
19. Zheng Z, Ma H, Lyu MR, King I (2009) WSRec: a collaborative filtering based web service recommendation system. In: Proceedings of the IEEE international conference on web services (ICWS 09). doi:10.1109/ICWS.2009.30
20. Jiang Y, Liu J, Tang M, Liu XF (2011) An effective Web service recommendation method based on personalized collaborative filtering. In: Proceedings IEEE international conference on web services (ICWS 11). doi:10.1109/ICWS.2011.38
21. Zhang L, Zhang B, Liu Y, Gao Y, Zhu Z (2010) A web service QoS prediction approach based on collaborative filtering. In: Proceedings IEEE Asia-Pacific services computing conference. doi:10.1109/APSCC.2010.43
22. Chen X, Liu X, Huang Z, Sun H (2010) RegionKNN: a scalable hybrid collaborative filtering algorithm for personalized Web service recommendation. In: Proceedings IEEE international conference on web services (ICWS 10). doi:10.1109/ICWS.2010.27
23. Tang Mingdong, Jiang Yechun, Liu Jianxun, Liu Xiaoqing (2012) Location-aware collaborative filtering for QoS-based service recommendation. IEEE international conference on web services (ICWS). doi:10.1109/ICWS.2012.61
24. Chee SHS et al (2001) Rectree: an efficient collaborative filtering method. In: Data warehousing and knowledge discovery, pp141–151
25. Sarwar B, Karypis G, Konstan J, Riedl J (2001) Itembased collaborative filtering recommendation algorithms. In WWW 2001. doi:10.1145/371920.372071
26. Silic M, Delac G, Krka I et al. (2013) Scalable and accurate prediction of availability of atomic web services. IEEE Trans Serv Comput
27. McLaughlin MR, Herlocker JL (2004) A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In SIGIR. doi:10.1145/1008992.1009050
28. Goldberg K, Roeder T, Gupta D, Perkins C (2001) Eigentaste: a constant time collaborative filtering algorithm. Inf Retr 4(2):133C151. doi:10.1023/A:1011419012209
29. Schafer JB et al (2007) Collaborative filtering recommender systems. The adaptive web, Springer, Berlin, Heidelberg, pp 291–324
30. Deshpande M, Karypis G (2004) Item-based top-n recommendation. ACM Trans Inf Syst 22:143–177. doi:10.1145/963770.963776