University of Huddersfield Repository

Qi, Lianyong, Xiang, Haolong, Dou, Wanchun, Bhuiyan, Md Zakirul Alam, Qin, Yongrui and Zhang, Xuyun

Privacy-preserving distributed service recommendation based on locality-sensitive hashing

Original Citation


This version is available at http://eprints.hud.ac.uk/id/eprint/31929/

The University Repository is a digital collection of the research output of the University, available on Open Access. Copyright and Moral Rights for the items on this site are retained by the individual author and/or other copyright owners. Users may access full items free of charge; copies of full text items generally can be reproduced, displayed or performed and given to third parties in any format or medium for personal research or study, educational or not-for-profit purposes without prior permission or charge, provided:

- The authors, title and full bibliographic details is credited in any copy;
- A hyperlink and/or URL is included for the original metadata page; and
- The content is not changed in any way.

For more information, including our policy and submission procedure, please contact the Repository Team at: E.mailbox@hud.ac.uk.

http://eprints.hud.ac.uk/
Privacy-preserving Distributed Service Recommendation based on Locality-Sensitive Hashing

Abstract—With the advent of IoT (Internet of Things) age, considerable web services are emerging rapidly in service communities, which places a heavy burden on the target users’ service selection decisions. In this situation, various techniques, e.g., collaborative filtering (i.e., CF) is introduced in service recommendation to alleviate the service selection burden. However, traditional CF-based service recommendation approaches often assume that the historical user-service quality data is centralized, while neglect the distributed recommendation situation. Generally, distributed service recommendation involves inevitable message communication among different parties and hence, brings challenging efficiency and privacy concerns. In view of this challenge, a novel privacy-preserving distributed service recommendation approach based on Locality-Sensitive Hashing (LSH), i.e., DistSR<sub>LSH</sub>, is put forward in this paper. Through LSH, DistSR<sub>LSH</sub> can achieve a good tradeoff among service recommendation accuracy, privacy-preservation and efficiency in distributed environment. Finally, through a set of experiments deployed on WS-DREAM dataset, we validate the feasibility of our proposal in handling distributed service recommendation problems.

Keywords—distributed service recommendation; privacy; efficiency; locality-sensitive hashing; collaborative filtering

I. INTRODUCTION

With the advent of IoT (Internet of Things) age, a great number of web services are emerging rapidly in service communities [1]. The ever-increasing services available on the web, on one hand, provide abundant alternatives for target users’ various service requirements, on the other hand, place a heavy burden on the target users’ service selection decisions especially when many service candidates share same or similar functionalities [2-3].

In order to alleviate the service selection burden of target users, various service recommendation techniques are brought forth in the last decade, e.g., widely adopted collaborative filtering (i.e., CF)-based recommendation [4-7]. Through considering the historical user-service quality data, traditional CF-based recommendation approaches (including memory-based CF, model-based CF and hybrid CF) can predict target users’ personalized preferences and further make accurate service recommendation.

However, existing CF-based service recommendation approaches often assume that the historical user-service quality data is centralized. Thus, the overall quality data generated from historical user-service invocations could be regarded as known already for subsequent service recommendation. While actually, the historical user-service quality data is sometimes not centralized, but distributed. For example, user A invoked web service WS<sub>A</sub> from Amazon, while user B invoked WS<sub>B</sub> from IBM.

In this distributed situation, two major challenges are raised. First, due to the privacy concern, neither Amazon nor IBM is willing to reveal the inner user-service quality data to each other, which makes it a difficult task to calculate the user similarity between A and B so as to make further recommendation. Second, due to the inevitable message communication between two distributed parties, i.e., Amazon and IBM, the service recommendation process if often time-consuming and cannot satisfy the target users’ quick response requirements.

In view of the above two challenges, we introduce the Locality-Sensitive Hashing (LSH) technique in service recommendation, and further put forward a novel privacy-preserving distributed service recommendation approach based on LSH, i.e., DistSR<sub>LSH</sub>. With the unique nature of LSH, DistSR<sub>LSH</sub> can achieve a good tradeoff among service recommendation accuracy, privacy-preservation and efficiency in distributed environment.

Generally, the contributions of our paper are three-fold.

(1) To the best of our knowledge, few existing works considered the service recommendation problem in distributed environment. We recognize the substantial significance of distributed service recommendation and specify the problem formally.

(2) We employ Locality-Sensitive Hashing technique to aid the distributed service recommendation, so as to achieve a good tradeoff among recommendation accuracy, privacy-preservation and efficiency.
which blocks the automatic service recommendation heavily. The abstract the representative feature tags of web services, besides, it becomes a challenging task to automatically methods often suffer from the over-specification problem providers. The vocabulary gap between mashup developers and service [5], the authors leverage mashup descriptions and structures in order to avoid the possible fake description of service tag, in similarity for content-based service recommendation. In applications, in [3], web service domain ontology is constructed by analyzing web service descriptions (e.g., WSDL and free text descriptors). Similarly, in [4], through analyzing the tree structure composed of synonyms and original meaning in semantic dictionaries such as HowNet and WordNet, the authors calculate the semantic distance between two services, and further obtain their semantic similarity for content-based service recommendation. In order to avoid the possible fake description of service tag, in [5], the authors leverage mashup descriptions and structures to discover important word features of services and bridge the vocabulary gap between mashup developers and service providers. However, the above content-based recommendation methods often suffer from the over-specification problem [6]. Besides, it becomes a challenging task to automatically abstract the representative feature tags of web services, which blocks the automatic service recommendation heavily.

B. CF-based Service Recommendation

Different from the content-based recommendation, CF-based recommendation works based on the past user-service invocation records. Generally, two categories are available: memory-based CF and model-based CF.

(1) memory-based CF

A comprehensive service recommendation approach $WSRec$ is put forward in [7], which combines user-based and item-based CF together. As service quality heavily depends on service invocation time, in order to make accurate service recommendation, a time-aware recommendation approach is brought forth in [8]. Similarly, location-aware service recommendation is performed in both [9] and [10], as geographically close users often experience similar service quality when they invoke an identical web service. Besides, different users hold distinct preferences, which also play an important role in service recommendation. Therefore, to make personalized recommendation, work [11] improves CF-based recommendation approach by integrating user preferences. Generally, the above memory-based CF recommendation approaches are easy-to-explain and effective when there is a great deal of available historical user-service quality data. However, memory-based CF approaches suffer from the scalability problem heavily, which means that the recommendation efficiency is often low when historical quality data is updated frequently. Beside, only few works (e.g., [12]) consider the privacy concern in recommendation. Third, the above approaches all assume that the historical user-service quality data is centralized, while neglect the distributed situations.

(2) model-based CF

Model-based CF approaches utilize the historical user-service quality data to build a recommendation model offline, and then make online recommendation based on the derived model. There are some classic model-based CF recommendation approaches, e.g., Matrix Factorization-based approaches [13], LDA-based approaches [14] and clustering-based approaches [15]. Generally, the above model-based CF approaches are efficient as the recommendation model could be trained offline. However, few works consider the privacy protection problem in recommendation process. Besides, similar to the memory-based CF approaches, the above model-based CF approaches are inappropriate to handle the distributed service recommendation.

With the above analyses, we can conclude that existing research works fall short in handling the distributed and privacy-preserving service recommendation problems. In view of the above shortcoming, a novel privacy-preserving distributed service recommendation approach named $DistSR_{LSH}$ is put forward in this paper, which will be specified in more detail in the next section.
III. Motivation

Here, we utilize the example in Fig.1 to motivate our paper. Concretely, target user \( u_1 \) invokes web services \( \{w_{31}, \ldots, w_{3n}\} \) from Amazon, user \( u_2 \) invokes web services \( \{w_{31}, \ldots, w_{3n}\} \) from Microsoft, and user \( u_1 \) invokes web services \( \{w_{31}, \ldots, w_{3n}\} \) from IBM. Then according to CF-based recommendation approach (e.g., user-based CF), the first step is to calculate the user similarity \( \text{sim}(u_1, u_2) \) and \( \text{sim}(u_1, u_3) \). However, the above similarity calculation process faces two major challenges:

1. As the historical user-service quality data is distributed on different platforms, Microsoft and IBM are often not willing to open their observed quality data to Amazon (here, target user \( u_1 \) is on Amazon platform), due to the privacy concern.

2. When the number of users or the number of services is large, the similarity calculation process may take a huge amount of time, as message communication is inevitable among Amazon, Microsoft and IBM; this means that the recommendation efficiency is often low and cannot satisfy the target users’ quick response requirements.

In view of the above two challenges, an efficient and privacy-preserving distributed recommendation approach named \( \text{DistSR}_{\text{LSH}} \) is introduced in the next section.

IV. Problem Formalization

In this paper, we focus on the distributed service recommendation. To facilitate the following discussions, we first formalize the distributed service recommendation problem as a four-tuple \( \text{DistSR}(PF; U, WS, q) \), where

1. \( PF = \{pf_1, \ldots, pf_z\} \): \( pf_k \) (\( 1 \leq k \leq z \)) denotes \( k \)-th distributed service platform; e.g., \( z = 3 \) holds in Fig.1.

2. \( U = \{U_1, \ldots, U_t\} \): \( U_k \) (\( 1 \leq k \leq z \)) denotes the user set corresponding to distributed platform \( pf_k \). Concretely, \( U_k = \{u_{k1}, \ldots, u_{km}\} \): \( u_{kl} \) (\( 1 \leq l \leq m \)) denotes \( l \)-th user on platform \( pf_k \) .

(3) \( WS = \{w_{31}, \ldots, w_{3n}\} \): \( w_{3j} \) (\( 1 \leq j \leq n \)) denotes the \( j \)-th web service. Here, to ease the following discussions, we assume that the services on every distributed platform \( pf_1, \ldots, pf_z \) are the same. For example, \( n_3 = n_2 = n_1 \) holds in Fig.1. Note that if a user did not invoke a service, then the corresponding user-service quality data is null.

4. \( q \) is a user-concerned quality dimension of web services, e.g., response time. For simplicity, we only consider a quality dimension in the following discussions.

V. A Privacy-Preserving Distributed Service Recommendation Approach: \( \text{DistSR}_{\text{LSH}} \)

In this section, we introduce the details of our proposed distributed service recommendation approach \( \text{DistSR}_{\text{LSH}} \). Concretely, we introduce the Locality-Sensitive Hashing technique briefly in subsection 5.A; afterwards, in subsection 5.B, we introduce the concrete steps of LSH-based recommendation approach \( \text{DistSR}_{\text{LSH}} \).

A. Locality-Sensitive Hashing

Locality-Sensitive Hashing, i.e., LSH was put forward by Alex Andoni in 1999 [16] and has been proven to be an effective technique to deal with many distributed applications, e.g., distributed information retrieval. Next, we introduce the unique property of LSH.

The main idea of LSH is: select a specific hashing function (or a hashing function family) so that (1) for two neighboring data points in original data space, they are still neighbors after hashing with a large probability (2) for two non-neighboring data points in original data space, they are still non-neighboring after hashing with a large probability.

A hashing function that satisfies the above two conditions are called a LSH function. More formally, a hashing function \( h(\cdot) \) is a LSH function iif the following conditions (1) and (2) hold, where \( x \) and \( y \) are two data points in original data space, \( d(x, y) \) denotes the distance between \( x \) and \( y \), \( h(x) \) is the hashing value of \( x \) after projection based on hashing function \( h(\cdot) \), \( P(X) \) represents the probability that condition \( X \) holds, \( \{d_1, d_2, p_1, p_2\} \) are a set of thresholds. If condition (1) and (2) hold simultaneously, then hashing function \( h(x) \) is a qualified LSH function and called \( (d_1, d_2, p_1, p_2) \)-sensitive.

If \( d(x, y) \leq d_1 \), then \( P(h(x) = h(y)) \geq p_1 \) \hspace{1cm} (1)

If \( d(x, y) \geq d_2 \), then \( P(h(x) = h(y)) \leq p_2 \) \hspace{1cm} (2)

Then through a LSH function \( h(\cdot) \) (or a LSH function family, see Fig.2), all the \( L \) data points \( \{x_1, \ldots, x_L\} \) in original data space could be projected into a set of buckets \( b_1, \ldots, b_L \) where each bucket \( b_i \) (\( 1 \leq i \leq L \) ) only contains \( l_i \) (\( l_i \ll L \) ) neighboring data points. Thus, if a target user wants to find the similar neighbors of original input \( X \) we can calculate \( h(X) \) and further find the bucket (assume \( b_i \) ) corresponding to \( h(X) \). Then according to the unique property of LSH, all the \( l_i \) data points in bucket \( b_i \) are similar neighbors of \( X \) with a large probability. Thus, the searching
space is reduced from \( L \) to \( l; \) as \( l; \ll L \) holds, the searching efficiency is improved significantly. Besides, through hashing projection, the privacy information of data points in original data space is transparent to the target users; for example, the target user in Fig.2 only know the hashing value \( h(x); \) but does not know the original value \( x. \) In this way, the data privacy is protected. This is the reason why LSH could be recruited for efficient and privacy-preserving distributed business applications. To facilitate the understanding of readers, the symbols recruited in this paper are specified in Table 1.

![Figure 2. LSH-based service recommendation process](image)

### Table 1. Symbol specifications

<table>
<thead>
<tr>
<th>symbol</th>
<th>specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z )</td>
<td>number of distributed service platforms</td>
</tr>
<tr>
<td>( m )</td>
<td>number of users in each service platform</td>
</tr>
<tr>
<td>( n )</td>
<td>number of web services (we assume services in different platforms are the same)</td>
</tr>
<tr>
<td>( q )</td>
<td>a quality dimension of web services</td>
</tr>
<tr>
<td>( d(\cdot) )</td>
<td>distance between two points</td>
</tr>
<tr>
<td>( P(\cdot) )</td>
<td>probability</td>
</tr>
<tr>
<td>( d_1, d_2, p_1, p_2 )</td>
<td>threshold</td>
</tr>
<tr>
<td>( h(\cdot) )</td>
<td>a LSH function</td>
</tr>
<tr>
<td>( H(\cdot) )</td>
<td>LSH function family</td>
</tr>
<tr>
<td>( L )</td>
<td>number of data points (or users); ( L = z * m ) holds here</td>
</tr>
<tr>
<td>( b_1, \ldots, b_l )</td>
<td>buckets in a hashing table</td>
</tr>
<tr>
<td>( l_i )</td>
<td>number of data points (or users) in bucket ( b_i )</td>
</tr>
<tr>
<td>( X )</td>
<td>input or profile of a target user</td>
</tr>
<tr>
<td>( T )</td>
<td>number of LSH tables</td>
</tr>
<tr>
<td>( r )</td>
<td>number of LSH functions in each LSH table</td>
</tr>
</tbody>
</table>

### B. DistSr_LSH-

**LSH-based Distributed Service Recommendation**

Generally, our proposed distributed web service recommendation approach \( \text{DistSr}_LSH \) (essentially a kind of improved user-based CF) consists of three steps (see Fig.2), each of which is generalized in Fig.3.

**Step 1: building user index offline.** Select a LSH function family to project the users on distributed platforms into corresponding buckets (the bucket No. is the user index) offline, based on the historical user-service quality data.

**Step 2: online neighbor finding.** According to the selected LSH function family, target user is projected into a bucket; then all the users in the bucket are considered as the target user’s similar neighbors (with a large probability).

**Step 3: Top-K service recommendation.** According to the target user’s neighbors derived in Step 2, predict target user’s quality over never-invoked services, and return the Top-K services.

![Figure 3. Three steps of distributed service recommendation approach DistSr_LSH](image)
Thus through LSH function in (3), user \( u \) is hashed into a binary value of 0 or 1. As indicated in (1) and (2), LSH is essentially a probability-based approach; therefore, the more hashing functions or hashing tables are recruited, the more accurate similarity is obtained. So in order to make accurate service recommendation, multiple LSH functions or tables are necessary. Concretely, assume there are \( T \) LSH tables, each of which consists of \( r \) LSH functions. Then for each LSH table, an \( r \)-dimensional vector (i.e., user index in the table) \( H(u) = (h_1(u), \ldots, h_r(u)) \) is obtained for user \( u \) after LSH, and each element in \( H(u) \) is equal to 0 or 1. Furthermore, two users \( u_1 \) and \( u_2 \) are hashed into an identical bucket after LSH, iff condition in (4) holds. Namely, if there is at least one LSH table (totally \( T \) tables) where the indexes of \( u_1 \) and \( u_2 \) are equal, then \( u_1 \) and \( u_2 \) could be regarded as similar neighbors. In this way, we can build index for any user distributed on different platforms, in an online manner.

\[ \exists x, \text{satisfy } H(u_x) = H(u_x') \quad (x \in \{1, \ldots, T\}) \quad (4) \]

**Step 2: online neighbor finding.**

In Step 1, we have built index for each user offline based on LSH. Next, for a target user \( u_{\text{target}} \), we can find his/her approximate neighbors online, whose major process is as follows: first, calculate index for \( u_{\text{target}} \) based on the adopted LSH function family; second, find the bucket corresponding to the index for \( u_{\text{target}} \); third, all the users in the bucket are regarded as similar neighbors of \( u_{\text{target}} \) with a large probability.

**Step 3: Top-K service recommendation.**

Next, we utilize the similar neighbors (derived in Step 2) of target user \( u_{\text{target}} \) to make service recommendation. Concretely, for each service \( ws \) never invoked by \( u_{\text{target}} \), we predict its quality \( q \) by \( u_{\text{target}} \) i.e., \( ws, q_{\text{target}} \) based on (5), where set \( NB \) denotes \( u_{\text{target}} \)'s neighbors derived in Step 2, \( w, q \) denotes \( ws \)’ quality over \( q \) observed by \( u \). Finally, we rank all the services (never invoked by \( u_{\text{target}} \)) by the predicted quality in (5) and select the Top-K services as the final recommendation results.

\[
ws, q_{\text{target}} = \frac{1}{|NB|} \sum_{u \in NB} ws, q_u \tag{5}
\]

Through the above three steps, we can finish the distributed service recommendation process and finally recommend \( K \) services to the target user. More formally, our proposal could be specified by pseudo code as below.

**Algorithm: DistSR_{LSH}**

**Inputs:** \( PF = \{p_1, \ldots, p_f\} \): distributed platforms \( U_f = \{u_{1f}, \ldots, u_{nf}\} \): user set for platform \( p_f \) \( WS = \{ws_1, \ldots, ws_n\} \): web service set \( q \): a quality dimension of web services \( u_{\text{target}} \): target user requesting recommended services

**Output:** \( \text{Rec}_\text{Ser}_\text{Set} \): service set recommended to \( u_{\text{target}} \)

/* Step 1: building user index offline */
for \( i = 1 \) to \( T \) do // T LSH tables
for \( j = 1 \) to \( z \)
for \( k = 1 \) to \( m \) do
\[ H(u_{ijk}) = (h_{1,w}, \ldots, h_{r,w}(u_{ijk})) \]
for \( j = 1 \) to \( r \) do // LSH functions
for \( k = 1 \) to \( n \) do // n-dimensional hashing vector
\[ h_{ijk} = \text{random} [-1, 1] \]
end for
if \( u_{ijk} \cdot h_{ijk} > 0 \) // dot product
then \( h_{ijk}(u_{ijk}) = 1 \)
else \( h_{ijk}(u_{ijk}) = 0 \)
end if
end for
end for
end for

/* Step 2: online neighbor finding */
set \( NB = \emptyset \) // neighbor set of \( u_{\text{target}} \)
for \( i = 1 \) to \( T \) do
\[ H(u_{\text{target}}) = (h_{1,w}(u_{\text{target}}), \ldots, h_{r,w}(u_{\text{target}})) \]
for \( j = 1 \) to \( r \) do
if \( u_{\text{target}} \cdot h_{ijk} > 0 \)
then \( h_{ijk}(u_{\text{target}}) = 1 \)
else \( h_{ijk}(u_{\text{target}}) = 0 \)
end if
end for
find bucket with index \( H(u_{\text{target}}) \) and put its users in \( NB \)
end for

/* Step 3: Top-K service recommendation */
for \( j = 1 \) to \( n \) do
if \( ws, q_{\text{target}} = 0 \) // \( u_{\text{target}} \) never invoked \( ws \) before
then \( \text{count} = 0 \)
for \( i = 1 \) to \( |NB| \) do // all neighbors of \( u_{\text{target}} \)
if \( ws, q_{ij} \neq 0 \) // i-th neighbor invoked \( ws \)
then \( \text{count}++ \)
\[ ws, q_{\text{target}} = ws, q_{\text{target}} + ws, q_{ij} \]
end if
end for
\[ ws, q_{\text{target}} = ws, q_{\text{target}} / \text{count} \]
end if
end for
put Top-K services with highest \( q \) into set \( \text{Rec}_\text{Ser}_\text{Set} \)
return \( \text{Rec}_\text{Ser}_\text{Set} \) to \( u_{\text{target}} \)

**VI. EXPERIMENT**

In this section, a set of experiments are conducted to validate the feasibility of our proposed DistSR_{LSH} approach, when dealing with the privacy-preserving distributed web service recommendation problems. The experiments are based on a real web service quality dataset WS-DREAM [19] which describes real-world QoS evaluation results from 339 users on 5825 Web services. We randomly divide the 339
users into 10 parts so as to simulate the distributed service recommendation scenario (in this paper, we mainly focus on the service recommendation efficiency and privacy concerns, so the user division manner does not affect the final experiment results). Besides, we consider the user-service response-time dimension, and 90% of the existing response-time data in WS-DREAM is removed so that we can predict the missing response-time data and make further recommendation. In each recommendation process, only Top-3 services are generated.

In order to validate the feasibility of our proposal in terms of distributed recommendation efficiency, accuracy and privacy-preservation, we test the following three criteria, respectively:

1. **time cost**: consumed time for generating service recommendation results.
2. **MAE** (Mean Absolute Error): average difference between predicted quality and real quality of recommended services.
3. **privacy-preservation**: confusion degree between real user-service quality and its index after LSH hashing.

Besides, we compare our proposed DistSR$_{LSH}$ approach with four related approaches: UPCC [20], IPCC [21], TLACF [9] (clustering-based approach, parameter flag = 0 and $d = 0$) and WSWalker [22] (random walk-based approach, parameter $\varepsilon = 0.0001$, max-depth = 6). The experiments were conducted on a Dell laptop with 2.80 GHz processors and 2.0 GB RAM. The machine is running under Windows XP and JAVA 1.5. Each experiment was carried out 10 times and the average experiment results were adopted finally.

A. Experiment Results and Analyses

Concretely, the following five profiles are tested and compared in our experiments. Here, as Table 1 indicates, $L$ and $n$ denote the number of users and number of web services, respectively, $T$ and $r$ denote the number of LSH tables and number of hashing functions in each LSH table, respectively.

**Profile1: efficiency comparison among five approaches**

In this profile, we test the recommendation efficiency of five approaches. The recruited parameters are set as below: $L$ is varied from 50 to 300, $n$ is varied from 1000 to 5000, $T = r = 10$ holds. The concrete experiment results are demonstrated in Fig.4.

In Fig.4(a), $n = 5000$ holds. The experiment results indicate that the time costs of five approaches all increase with the growth of $m$, while our proposed DistSR$_{LSH}$ approach outperforms the other four ones as most work (i.e., user index building) in DistSR$_{LSH}$ could be done offline, and the candidate space for similar neighbors of a target user is reduced significantly with the help of unique nature of LSH. Thus, the recommendation efficiency is improved significantly. Similar results could be observed from Fig.4(b), which is not explained repeatedly.

![Figure 4. Recommendation efficiency comparison](image4)

![Figure 5. Recommendation accuracy comparison](image5)
Profile 2: accuracy comparison among five approaches

In this profile, we test the accuracy (i.e., MAE, the smaller the better) of five recommendation approaches. The experiment parameters are set as follows: \( L \) is varied from 50 to 300, \( n \) is varied from 1000 to 5000, and \( T = r = 10 \) holds. The experiment results are shown in Fig. 5.

Concretely, in Fig.5(a), \( n = 5000 \) holds; and the experiment results demonstrate that the recommendation accuracies of five approaches do not change significantly with the growth of \( m \) (except for IPCC approach), as only Top-3 services are recommended to the target user in each recommendation approach; besides, our proposed DistSR\(_{LSH}\) approach outperforms the rest four ones in terms of recommendation accuracy, which is due to the inherent nature of our adopted LSH technique (i.e., two neighboring users are projected into the same bucket after LSH hashing with a large probability).

In Fig.5(b), \( m = 300 \) holds; and the experiment results show that the recommendation accuracy of our DistSR\(_{LSH}\) approach sometimes outperforms the other four ones (e.g., when \( n = 3000 \) or 5000); while when \( n = 1000, 2000 \) or 4000, DistSR\(_{LSH}\) does not perform very well in terms of recommendation accuracy, which is due to the fact that our adopted LSH technique is actually a probability-based technique and hence, cannot always guarantee to achieve an optimal service recommendation result. However, as Fig.5(b) shows, the accuracy of DistSR\(_{LSH}\) approach is still acceptable in most cases.

Profile 3: privacy-preservation of DistSR\(_{LSH}\) w.r.t. \( T \) and \( r \)

The four related recommendation approaches, i.e., UPCC, IPCC, TLACF and WSWalker do not consider the privacy protection; therefore, in this profile, we only test the privacy-preservation effect of our proposed DistSR\(_{LSH}\) approach. According to DistSR\(_{LSH}\), the original user-service quality data are hashed into different buckets based on the calculated LSH index. Therefore, in this profile, we utilize the bucket density (i.e., the number of derived similar neighbors) to approximately represent the privacy-preservation effect (a larger density often means better privacy-preservation effect). The experiment parameters are set as follows: \( L = 300, n = 5000, T \) is varied from 10 to 20, \( r \) is varied from 2 to 12. The experiment results are shown in Fig. 6.

As Fig.6 shows, when the number of hashing functions in each LSH table, i.e., \( r \) is small, many users are projected into the same bucket as the target user, namely many candidate neighbors are obtained. While with the growth of \( r \), the number of neighbors is reduced significantly, as the neighbor “criteria” becomes stricter.

Profile 4: accuracy of DistSR\(_{LSH}\) w.r.t. \( T \) and \( r \)

In this profile, we test the service recommendation accuracy of DistSR\(_{LSH}\) with respect to \( T \) and \( r \). The experiment parameters are set as follows: \( L = 300, n = 5000, T \) is varied from 10 to 20, \( r \) is varied from 2 to 12. The experiment results are shown in Fig. 7.

As Fig.7 shows, the MAE values of DistSR\(_{LSH}\) decrease approximately with the growth of \( r \); this is because a larger \( r \) means stricter filtering condition for similar neighbors and hence, those “really similar” neighbors of a target user could be obtained finally.

Profile 5: efficiency of DistSR\(_{LSH}\) w.r.t. \( T \) and \( r \)

In this profile, we test the efficiency of DistSR\(_{LSH}\) with respect to \( T \) and \( r \). Here, \( L = 300, n = 5000, T \) is varied from 10 to 20, \( r \) is varied from 2 to 12. The experiment results are shown in Fig. 8. As Fig.8 shows, the time costs of DistSR\(_{LSH}\) decrease with the growth of \( r \); this is because when \( r \) grows, the filtering condition becomes stricter and only a few “really similar” neighbors are returned and hence, the recommendation efficiency is improved.
VII. CONCLUSIONS

In this paper, we put forward a novel privacy-preserving service recommendation approach based on Locality-Sensitive Hashing, i.e., DistSRSSH, to handle the distributed recommendation problem. Through Locality-Sensitive Hashing, candidate space of similar neighbors of a target user could be reduced significantly, so that the recommendation efficiency is improved. Besides, for distributed users, their invoked service quality information is transparent to the target users through hashing technique, which protects user privacy very well. Finally, we validate the feasibility of our proposal through a set of experiments deployed on well-known WS-DREAM dataset. The experiment results demonstrate that our proposed DistSRSSH approach can achieve a good tradeoff among service recommendation accuracy, privacy-preservation and efficiency in distributed environment.

In the future, we will investigate the distributed service recommendation problems with multiple quality dimensions. Besides, service quality is not stable, but dynamic; therefore, we will study the dynamic quality-aware distributed service recommendation problems in the future.

ACKNOWLEDGMENT

This paper is partially supported by Natural Science Foundation of China (No. 61402258, No. 61672276), Open Project of State Key Laboratory for Novel Software Technology (No. KFKT2016B22).

REFERENCES