Integrated Content and Network-Based Service Clustering and Web APIs Recommendation for Mashup Development

Buqing Cao, *Member, IEEE*, Xiaoqing (Frank) Liu, *Member, IEEE*, MD Mahfuzer Rahman, Bing Li, *Member, IEEE*, Jianxun Liu, and Mingdong Tang

Abstract—The rapid growth in the number and diversity of Web APIs, coupled with the myriad of functionally similar Web APIs, makes it difficult to find most suitable Web APIs for users to accelerate and accomplish Mashup development. Even if the existing methods show improvements in Web APIs recommendation, it is still challenging to recommend Web APIs with high accuracy and good diversity. In this paper, we propose an integrated content and network-based service clustering and Web APIs recommendation method for Mashup development. This method, first develop a two-level topic model by using the relationship among Mashup services to mine the latent useful and novel topics for better service clustering accuracy. Moreover, based on the clustering results of Mashups, it designs a collaborative filtering (CF) based Web APIs recommendation algorithm. This algorithm, exploits the implicit co-invocation relationship between Web APIs inferred from the historical invocation history between Mashups clusters and the corresponding Web APIs, to recommend diverse Web APIs for each Mashups clusters. The method is expected to not only find much better matched Mashups with high accuracy, but also diversify the recommendation result of Web APIs with full coverage. Finally, based on a real-world dataset from ProgrammableWeb, we conduct a comprehensive evaluation to measure the performance of our method. Compared with existing methods, experimental results show that our method significantly improves the accuracy and diversity of recommendation results in terms of precision, recall, purity, entropy, DCG and HMD.

Index Terms—Service content, service network, two-level topic model, random walk, web APIs recommendation

1 Introduction

Recently, Mashup technology, which allows software developers to compose existing Web APIs to create value-added composite RESTful Web services, has emerged as a promising software development method in service-oriented environment [1]. Several online Mashups and Web APIs repositories have been established, such as ProgrammableWeb, myExperiment, and Biocatalogue. In these repositories, there are a large number of published Web APIs for providing external invocation, and users can compose various Web APIs with different functionalities to create Mashup for completing users' requirement [1].

To fulfill Mashup development, selecting most suitable Web APIs from the repositories for users is a difficult task. For example, ProgrammableWeb has published more than 16,968 Web APIs belong to more than 400 predefined categories also have been created with these services as to Februrary 2017. When a user wants to develop a Mashup application related to mobile, there are more than 1,850 Web APIs found by search engine of ProgrammableWeb. It is difficult for user to select the most suitable Web APIs from so many Web APIs with the functionality of mobile to complete Mashup development. Besides, we observed that some Web APIs returned by the search engine do not meet users' Mashup requirement.

To address the above problem, some researchers exploit service recommendation techniques to achieve effective service discovery [2], [3], [4], [5]. Their methods mainly include content-based [6], QoS-based [7], [8], [9], [10], structure-based [11], [12], and hybrid service recommendations [14], [15], [16], [17]. Most recently, few researchers incorporate service clustering technique into service recommendation [1], [18]. Especially in the large-scale services space, similar services in terms of functionality are grouped into clusters in order that services' search space can be expanded and more services can be discovered together. Even though the existing methods show improvements in service recommendation, there are two unaddressed problems which affect the performance of service recommendation.

The first problem is that few of existing methods perceive that service documents are related to each other and none of them uses their relationship information for Mashup service clustering. Several researchers exploit K-Means algorithm

- *B. Cao is with the School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China and the Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701. E-mail: buqingcao@gmail.com.*
- *X. Liu and M.M. Rahmanare with the Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701. E-mail: {frankliu, mmr014}@uark.edu.*
- *B. Li is with the International School of Software, Wuhan University, Wuhan 430072, China. E-mail: bingli@whu.edu.cn.*
- *J. Liu and M. Tang are with the Department of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China. E-mail: {ljx529, tangmingdong}@gmail.com.*

Manuscript received 12 June 2016; revised 28 Feb. 2017; accepted 16 Mar. 2017. Date of publication 22 Mar. 2017; date of current version 12 Feb. 2020. Digital Object Identifier no. 10.1109/TSC.2017.2686390

based on term vector space model TF-IDF [18] or extended K-Means algorithm based on topic model LDA [1] to cluster Mashups. A limitation of these works is that service documents were independently used to derive service vector space or individual topics for calculating the similarity among services and identify service clusters. Actually, Mashups are related to each other due to common Web APIs' invoking or tags' marking, and form a large service network based on their relationships [13]. The relationships among Mashups can be exploited to mine useful and novel topics for significantly improving accuracy of both clustering and recommendation.

The second problem is that most existing methods may produce a recommendation result with only a single focus, such popularity. It is desirable to return a set of diverse Web APIs in order to better cover the searching space of Web APIs, including both popular and unpopular ones. According to the statistical results in the ProgrammableWeb [5], the top 10 (200) popular Web APIs invocations in Mashups cover about 30.6 percent (99 percent) of all Web APIs invocations in Mashups. This indicates only a small portion of Web APIs is frequently used in Mashups, while most of them are actually with low usage or even not used. Since users usually consider the historical usage of Web APIs and tend to create new Mashups by using these popular Web APIs, which will result in the recommendation results only focusing on the popluar Web APIs. Consequently, those unpopular Web APIs which are really suitable for users' Mashup requirement may not be found [18]. Actually, the implicit co-invocation records between Web APIs in common historical Mashups can be used to predict usage probability of unpopular Web APIs in Mashups. The prediction results can be used to diversify the Web APIs recommendation.

Inspired by above approaches, the goal of this paper is to use the relationship among Mashup service documents and the implicit co-invocation between Web APIs to obtain more accurate and diverse Web APIs recommendation results for users' Mashup development. The contributions are summarized as follows:

- We present a novel two-level topic clustering model for mining more useful topics from the relationship among Mashups, and modeling Mashup services in terms of both their content and network for effective representation of functional features of Mashup services. In this model, we employ random walk process on the Mashup service network to identify novel topics from the linked Mashup documents at the network level and incorporate them into the topic probability distribution of original Mashup service documents at the content level.
- We use Jensen-Shannon (JS) divergence to calculate the similarity based on latent functional topics between Mashup service documents, and combine K-Means and Agnes algorithms to perform Mashup service clustering. Based on the clustering results of Mashups, we design a CF-based Web APIs recommendation algorithm, to explore the historical invocation history between Mashups clusters and the corresponding Web APIs, and derive the implicit co-invocation relationship among Web APIs to rank and recommend diverse Web APIs for users' Mashup requirement.
- We develop a real-world dataset from ProgrammableWeb, which can be accessed and downloaded in the URL http://49.123.0.60:8080/MashupNetwork2.0/dataset.jsp. We also conduct a set of experiments and experimental results show that our method achieves a significant improvement in terms of clustering accuracy (precision, recall, purity and entropy), recommendation accuracy (DCG) and diversity (HMD), compared with other existing approaches.

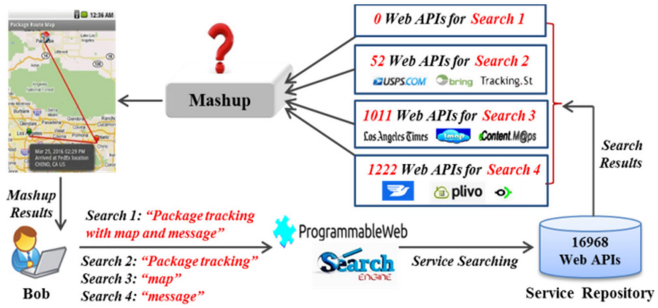

Fig. 1. Motivation example.

The rest of this paper is organized as follows: Section 2 introduces background. Section 3 describes the method. Section 4 reports experimental results. Section 5 shows the related works. Section 6 provides dicussions. Finally, conclusions and future work are presented in Section 7.

2 Background

2.1 Motivation Example

Suppose that a developer Bob wants to develop a Mashup application that can track a package with map display and message notification. He first enters his search requirement "*Package tracking with map and message*" in the search engine of ProgrammableWeb, it found 0 search results. It does not help at all. Then, he searches Web APIs of the target Mashup using other search requirements such as "*package tracking*", "*map*" and "*message*", and respectively obtains 52/ 1,011/ 1,222 search results, as illustrated in Fig. 1.

Specifically, by examining the 52 search results using keywords "*package tracking*", we have the following observations:

- They belong to 19 service categories with different functionalities.
- Many relevant Web APIs are missing. For example, several related services were not found, such as Canada-Post-Tracking, Australia-Post-Tracking.
- Many of them are irrelevant. They do not meet the requirement of Bob for Web APIs of his target Mashup, such as, Bower, GAMEhud, and VersaPay.

The same problems exist with other two search results of using keywords "*map*" and "*message*". As a result, it is very challenging to select proper Web APIs to compose them for the Mashup creation based on the above unsatisfactory search results with a large number of Web APIs. There are several main causes for the above problems. First, the manual, predefined categories in ProgrammableWeb are too rigid, incomplete, and vague [1]. An effective service

clustering method is needed and relationships between services should be used for service categorization. Second, the search engine of ProgrammableWeb neglects the diversity of Web APIs which can be used for novel Mashup development. For example, even if *Canada-Post-Tracking* may be unpopular in service repository, we can compose it with *Google Maps* and *MessageBird* to build a personalized Mashup application that can track a *Canada-Post* package with *Google Maps* display and *MessageBird* notification in Canada. Finally, the keyword-based searching technique used by ProgrammableWeb leads to many irrelevant search results. The historical combinations of Web APIs in common Mashups can be used to increase the relevance of service search.

2.2 Requirements and Challenges

To resolve the above problems, we identify the following major requirements in order to make effective Web APIs recommendation for Mashup development:

- *High clustering accuracy.* Mashup service clustering with high accuracy significantly improves the quality of Web APIs recommendation. Similar Mashup services in terms of functionality are grouped into clusters in order that they can be searched and discovered.
- *Diverse recommendation result.* A recommendation result can't include only popular Web APIs and other unpolpolar but useful Web APIs should be included too. A diverse Web APIs recommendation result would have a better coverage of Web APIs in Mashup development.
- *Good recommendation relevance.* A good recommendation method should recommend more relevant Web APIs and fewer irrelevant ones, particularly in the situations where the Mashup-Web API matrix is very sparse.

As discussed in the Section 1, the existing clustering-based Web APIs recommendation methods for Mashup creation cannot satisfy all the above three requirements at the same time. It is highly desirable for Web service search engine to classify Mashup services into clusters accurately and recommend Web APIs with good relevance and diversity for Mashup development. Our method will incorporate relationships among Mashup services and implicit combinatorial usage of Web APIs to achieve high-quality Web APIs recommendation. In this method, we focus on the following three key challenges:

- *Mashup service clustering.* How to exploit the relationship among Mashup services and integrate it with service content in order to classify Mashup services into various clusters with high accuracy for providing a basis for Web APIs recommendation for Mashup development?
- *Modeling and matching between Mashup requirement and Mashup clusters.* How to model and represent Mashup requirement and Mashup clusters, and map the Mashup requirement to the best suitable Mashup cluster for conducting subsequent Web APIs recommendation?
- *Web APIs recommendation.* How to explore the implicit combinatorial usage of Web APIs in common Mashup service clusters from their usage history to rank and recommend relevant and diverse Web APIs?

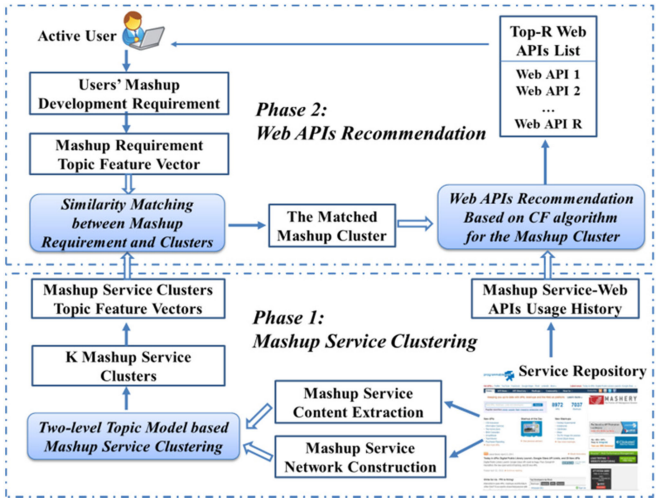

Fig. 2. Framework.

3 Method Overview

To address the above challenges, we have designed a framework of Mashup service clustering and Web APIs recommendation, as shown in Fig. 2.

The framework is composed of two phases: Mashup service clustering (Phase 1), and Web APIs recommendation (Phase 2). In phase 1, functional profile information of Mashup services is crawled and parsed from online service repository. The Mashup service-Web APIs usage history is obtained from the historical invocation times between Mashup services and Web APIs. The core process of the phase 1 is two-level topic model based Mashup service clustering in terms of both Mashup service content and Mashup service network. K Mashup service clusters are obtained by applying the two-level topic model to derive topics of Mashup services for more accurate service clustering. In phase 2, when an active user submits a Mashup development requirement with textual description, which is first converted a corresponding Mashup requirement topic feature vector by using LDA technology. Meanwhile, the same process is performed as for identification of K Mashup service clusters in phase 1, and obtains Mashup service clusters topic feature vectors. Then similarity based matching using JS divergence between Mashup requirement and clusters is used to identify a Mashup cluster with the highest similarity. Finally, a Web APIs recommendation based on CF algorithm is designed to recommend top-R Web APIs to the user. It uses historical invocation records between Mashups clusters and Web APIs to rank and recommend diverse Web APIs, including popluar and unpopular ones.

3.1 Service Content Extraction and Service Network Construction

We first crawl functional profile information of Mashup services from internet (including its category, name, description, Web APIs and tags) to build Mashup service documents. Then, we perform a preprocessing to extract

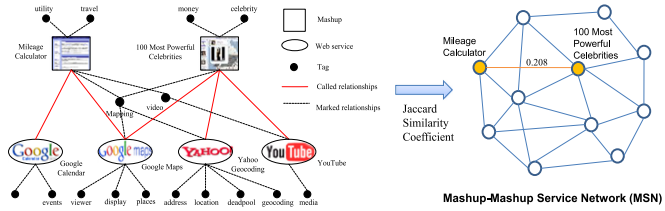

Fig. 3. An example of mashup service network.

feature vectors representing their content. The main steps of it include [18]: (1) build initial vector; (2) remove stop words; and (3) extract stemming. A preprocessed Mashup service document is represented as a content feature vector *MS(C, N, T, WA, T)*, where *C* is category, *N* is name, *T* is description text, *WA* is Web APIs, and *T* is tags.

A Mashup Service Network (*MSN*) is build to represent the relationships among Mashup services and facilitate latent functionality topics mining. The relationship among Mashup services can be described by an undirected network graph $MSN = (MS, E(MS_i, MS_j), W(MS_i, MS_j))$, where, *MS* is a preprocessed Mashup service nodes set; MS_i and MS_j represent any two Mashup service nodes in *MS*; $E(MS_i, MS_j)$ represents an undirected edge which connects service nodes; $W(MS_i, MS_j)$ is the weight of $E(MS_i, MS_j)$, which represents similarity of the relationship between MS_i and MS_j. Generally speaking, if two Mashup services (i.e., MS_i and MS_j) have same marking tags or commonly invoke same Web APIs, they are considered to be similar in functionality or belong to a same service category [19]. We use Jaccard similarity coefficient [19] to measure the similarity weight of edge in the *MSN*:

$$W\left(MS_i, MS_j\right) = \lambda_1 * \frac{\left|API(MS_i) \cap API\left(MS_j\right)\right|}{\left|API(MS_i) \cup API\left(MS_j\right)\right|} + \lambda_2 * \frac{\left|TAG(MS_i) \cap TAG\left(MS_j\right)\right|}{\left|TAG(MS_i) \cup TAG\left(MS_j\right)\right|} \quad (1)$$

Here $API(MS_i)$ and $API(MS_j)$ represent Web API sets, which are invoked by MS_i and MS_j respectively. $TAG(MS_i)$ and $TAG(MS_j)$ represent tags sets, which mark MS_i and MS_j respectively. λ_1 and λ_2 are users' preferences, $\lambda_1 + \lambda_2 = 1$. An example of the *MSN* is shown in Fig. 3. There are two Mashup services, i.e., *100 Most Powerful Celebrities* (MS_i) and *Mileage Calculator* (MS_j), and four Web APIs, i.e., *Yahoo Geocoding*, *Google Calendar*, *YouTube* and *Google Maps*, and sixteen tags. Based on the invoking and marking relationships, we construct a *MSN* and calculate its edge weights. As for this example, we set $\lambda_1 = \lambda_2 = 1/2$, $W\left(MS_i, MS_j\right) = 0.5 * 1/4 + 0.5 * 1/6 = 0.208$. The edge weight determines the scale and density of the *MSN*, which significantly affect the clustering performance. So, we set a threshold *T* for $W(MS_i, MS_j)$ to investigate its effect on clustering in the experiment section.

3.2 Two-Level Topic Model Based Service Clustering

We develop a two-level topic level to model Mashup services in terms of both their content and network, and derive useful topics. In this model, two random walk processes are employed to incorporate the topics at the network level into the topics at the content level. Based on the topic distribution results of Mashup services, we use JS divergence to calculate the similarity between Mashup services, and combine K-Means and Agnes algorithms to achieve Mashup service clustering.

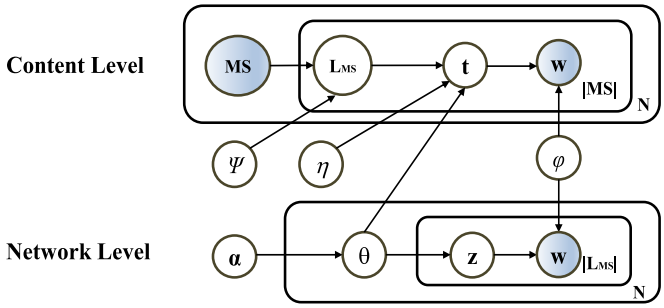

Fig. 4. A two-level topic model.

3.2.1 Two-Level Topic Model

Different from existing topic models [20], [21], the topics of Mashup service document is modeled at two levels: content and network. Our submodel at the content level incorporates original Mashup service documents for clustering. We construct their *MSN* from the original Mashup service documents, representing relationships of Mashup services based on directly and indirectly linked Mashup services from another submodel at the network level. The topics of linked Mashup services in the *MSN* at the network level are incorporated into the topics of original Mashup service documents at the content level. The final topics of each original Mashup service document include two parts: topics from itself and other "novel" topics from the linked Mashup services of its *MSN*. The novel topics is captured and derived by random walk process on its *MSN*. The probabilistic model is illustrated in Fig. 4.

In the Fig. 4, the generative process of each Mashup service document is as below:

1) The modeling performs the generative process at the network level for all linked Mashup services in the MSN of original Mashup service document *MS*. Where L_{MS} represents the linked Mashup services set of *MS*; $|L_{MS}| = \mathrm{N}$, shows that a Mashup service may potentially link all Mashup services (including itself).
   → *For each directly/indirectly linked Mashup service document* MS_j *of MS in* L_{MS}. *For the ith word in* MS_j*:*
      a. *Select a topic* z_{ji} *from the topic distribution of* MS_j, $p(z|MS_j, \theta_{MS_j})$, *where the distribution parameter* θ_{MS_j} *is gained from a Dirichlet distribution Dir(*α*).*
      b. *Select a word* w_{ji} *which follows the multinomial distribution* $p(w|z_{ji}, \varphi)$ *conditioned on the topic* z_{ji}.
2) The modeling performs the generative process at the content level for original Mashup service documents. Here, we set $MS = \{MS_1, MS_2, \ldots, MS_N\}$, i.e., $|MS| = N$, *N* is the number of original Mashup service documents.
   → *For each original Mashup service document* MS_s *in MS. For the* ith *word in* MS_s*:*
      a. *Select a linked Mashup service document* $L_{MS_{si}}$ *from* $p(L_{MS}|MS_s, \Psi)$, *a multinomial distribution on* MS_s.

b. *Select a topic t_{si} from $p(t|L_{MS_{si}}, \eta)$ of $L_{MS_{si}}$ at the network level, a multinomial distribution on $L_{MS_{si}}$.*
c. *Select a word w_{si} which follows the multinomial distribution $p(w|t_{si}, \varphi)$ conditioned on the topic t_{si}.*

As described in the above generative processes, t and z respectively represent the latent topics at the content level and network level. It is worth noting that, Ψ and η are two different coefficient matrix, and they represent how many the contents and topics of MS at the content level are from L_{MS} at the network level, respectively. The composition of Ψ and η models the topic distribution at the content level. Ψ is a N^*N Mashup service selection coefficient matrix, where $\Psi_{js} = P(L_{MS_{si}} = MS_j | MS_s)$, $1 \le j \le N$, $1 \le s \le N$, which represents the probability of MS_j at the network level will be incorporated into the MS_s at the content level. η is a L^*N topic selection coefficient matrix, where $\eta_{ij} = P(t_{si} = z_{ji} | L_{MS_{si}})$, which represents the topic selection probability to select the topic z_{ji} from the topic distribution of the $L_{MS_{si}}$ at the network level, and L is the number of different topics generated by MS_j. The detailed computation process of Ψ and η will be shown with details in the next section.

3.2.2 Random Walk in the Mashup Service Network

We propose two random walk processes to derive the topic distributions of each directly/indirectly linked Mashup service document at the network level.

First of all, we perform a link-level random walk on the *MSN* of MS_s to derive the matrix Ψ. For each MS_j in L_{MS}, we associate it with a link probability score from MS_s:

$$P(L_{MS_{si}} = MS_j | MS_s) = (1-\beta) * (I - \beta Q)^{-1} M \quad (2)$$

Here, β is one probability that random walk stops at the current node MS_j; $(1-\beta)$ is another probability that random walk jumps to any other nodes in the *MSN* of MS_s. β depends on the scale and density of the *MSN*, and we will investigate its effect on clustering in the experiment section. M is an initial probability distribution vector of all MS_j in the *MSN*, $M = [m_1, m_2, \ldots, m_N]$, $m_j = 1/L(MS_j)$. $L(MS_j)$ is a degree of the MS_j, representing how many nodes directly connect to MS_j in the *MSN*.

It is worth noting that Q is an adjacency matrix, i.e., $Q = [q_{ij}]_{N*N}$. q_{ij} is a transition probability random walk transfers from MS_i to MS_j. The value of q_{ij} is equal to $W(MS_i, MS_j)$ in the formula (1), i.e., $q_{ij} = W(MS_i, MS_j)$.

Through the above random walk process, the link probability scores of all MS_j in the *MSN* of each MS_s will be derived. For all original Mashup service documents, we repeat the above process and obtain their final link probability distribution to construct the matrix Ψ, i.e., $[\Psi_{js}]_{N*N} = [P(L_{si} = MS_j | MS_s)]_{N*N}$.

Second, we perform a topic-level random walk on the *MSN* of the MS_s to derive the matrix η. For MS_j in L_{MS}, we associate it with a topic probability score vector $P(MS_j, z)$, each of which is specific to topic z. Random walk is performed along with MS_j in the *MSN* within all the topics. For MS_i having a link to MS_j, we define two types of transition probabilities from MS_i to MS_j: topic-intra (sharing common topics) transition probability and topic-inter (across different topics) transition probability:

$$P(MS_j | MS_i, z_x) = \frac{1}{L(MS_i)} \quad (3)$$

$$P(MS_j, z_x | MS_i, z_y) = P(z_x | MS_j) * P(z_y | MS_i) \quad (4)$$

Here,

- $P(MS_j | MS_i, z_x)$—the topic-intra transition probability from MS_i to MS_j on the common topic z_x;
- $P(MS_j, z_x | MS_i, z_y)$—the topic-inter transition probability from MS_i to MS_j across different topics z_x and z_y;
- $L(MS_i)$—the degree of MS_i, representing how many nodes directly connect to MS_i;
- $P(z_x | MS_j)$—the probability of topic z_x generated by MS_j;
- $P(z_y | MS_i)$—the probability of topic z_y generated by MS_i.

Besides, we introduce a parameter γ to represent preference to topic-intra or topic-inter transition probability. Therefore, the random walk starting from MS_i will have a γ probability to access the common topics on MS_j and $(1-\gamma)$ probability to find across different topics on MS_j.

Though the above topic-level random walk process, we gain a topic probability score for MS_j on its topic z_x:

$$P(MS_j, z_x) = \beta \sum_{MS_i : MS_i \to MS_j} \Big[\gamma P(MS_j | MS_i, z_x) + (1-\gamma) \frac{1}{|T|} \sum_{y \ne x} P(MS_j, z_x | MS_i, z_y) \Big] + (1-\beta) \frac{1}{|D|} P(z_x | MS_j) \quad (5)$$

Here,

- $P(MS_j, z_x)$—the topic probability score of MS_j on the topic z_x;
- $P(z_x | MS_j)$—the probability of topic z_x generated by MS_j;
- $P(MS_j | MS_i, z_x)$—the transition probability from MS_i to MS_j on the common topic z_x;
- $P(MS_j, z_x | MS_i, z_y)$—the transition probability from MS_i to MS_j across different topics z_x and z_y;
- $|D|$—the number of Mashup service documents in the *MSN* of MS_s;
- $|T|$—the topic number generated by MS_j.

The parameter β in the above formula (5) is the same as the one described in the formula (2), which specifies the probability of random walk from MS_i to MS_j; $(1-\beta)$ is the probability of random walk which jumps to any other indirectly linked nodes in the *MSN* of MS_s. Therefore, for all topics generated by MS_j, we gain their topic probability scores by the formula (5) and use them to build a final topic probability score vector $P(MS_j, z)$, i.e., $P(MS_j, z) = [P(MS_j, z_1), P(MS_j, z_2), \ldots, P(MS_j, z_L)]^{\mathrm{T}}$, L is the number of topics generated by MS_j.

Similar to the construction of Ψ, we integrate the topic probability score vectors of all MS_j in the *MSN* of each MS_s to build the η, i.e., $[\eta_{ij}]_{L*N} = [P(t_{si} = z_{ji} | L_{MS_{si}})]_{L*N} = [P(MS_1, z), P(MS_2, z), \ldots, P(MS_N, z)]$.

TABLE 1
Topic Distributions of Mashup Services

	t_1	t_2	t_3
MS_1	0.31	0.14	0.27
MS_2	0.37	0.25	0.19

3.2.3 Mashup Service Clustering

For the topic distributions of all original Mashup service documents at the content level described in the above section, we use Kullback-Leibler (KL) and JS divergence [22] to calculate the similarity among Mashup services, and then combine K-Means and Agnes algorithms to perform service clustering. The topics of Mashup service are a simple mapping of its document vector space and represent its core content in order that the similarity among Mashup services can be calculated by using their topic probability distribution. Since the topic is a mixture distribution of word vector, the below KL divergence can be introduced as the similarity measurement:

$$D_{KL}\left(MS_i, MS_j\right) = \sum_{t=1}^{T} p_t ln \frac{p_t}{q_t} \tag{6}$$

Here,

- $D_{KL}(MS_i, MS_j)$—KL divergence between Mashup services MS_i and MS_j;
- t—a variable shows common topics in MS_i and MS_j;
- T—the total number of common topics in MS_i and MS_j;
- p_t—the probability of topic t in MS_i;
- q_t—the probability of topic t in MS_j.

Although the KL divergence is often used as a way of measuring the distance between topic probability distributions, it is not an effective metric in our application. It does not show the triangle inequality and is asymmetric, i.e., $D_{KL}(MS_i, MS_j) \neq D_{KL}(MS_j, MS_i)$. Since an interrelationship of two topics in MS_i and MS_j is symmetric, it is impractical to use the KL divergence to measure the similarity between Mashup services. The JS divergence is an improved, symmetrical version of KL divergence, which can exactly measure the divergence between topic semantics. Here, we below employ the JS divergence on top of KL divergence to measure the similarity between MS_i and MS_j. A smaller JS divergence means a higher similarity between any two Mashup services.

$$\begin{aligned} D_{JS}\left(MS_i, MS_j\right) = \frac{1}{2} & \left[D_{KL}\left(MS_i, \frac{MS_i + MS_j}{2}\right) \right. \\ & \left. + D_{KL}\left(MS_j, \frac{MS_i + MS_j}{2}\right) \right] \end{aligned} \tag{7}$$

Suppose the topic distribution probabilities of Mashup services MS_1 and MS_2 as the below Table 1 (t_1, t_2 and t_3 are common topics in MS_1 and MS_2), based on the formulas (6) and (7), we can gain the KL and JS divergences of them, i.e., $D_{KL}\left(MS_1, \frac{MS_1+MS_2}{2}\right) = 0.31 * ln\frac{0.31}{0.34} + 0.14 * ln\frac{0.14}{0.195} + 0.27 * ln\ \frac{0.27}{0.23} = -0.0317$, $D_{KL}\left(MS_2, \frac{MS_1+MS_2}{2}\right) = 0.37 * ln\frac{0.37}{0.34} + 0.25 * ln\frac{0.25}{0.195} + 0.19 * ln\ \frac{0.19}{0.23} = 0.0571$, $D_{JS}\left(MS_1, MS_2\right) = 0.5 * [D_{KL}(MS_1, \frac{MS_1+MS_2}{2}) + D_{KL}(MS_2, \frac{MS_1+MS_2}{2})] = 0.0254$.

Therefore, the similarity of any two Mashup service documents, denoted as $Sim(MS_i, MS_j)$, can be calculated by the above formula (7), i.e., $Sim\left(MS_i, MS_j\right) = D_{JS}\left(MS_i, MS_j\right)$. It is used as matrix's elements to construct a similarity matrix among all Mashup service documents for clustering. Similar to our previous work [22], K-means and Agnes algorithm are used to perform the below Mashup service clustering process in terms of their similarities.

(1) Build a Mashup service similarity matrix *MSim*, and an average similarity set ASS for each Mashup service;
(2) Rank Mashup services and select some of them with higher average similarity to build N' atom-clusters by K-Means algorithm. Build a Mashup clusters similarity matrix *CSim* between the N' atom-clusters;
(3) Use Agnes algorithm to conduct hierarchical clustering for the N' atom-clusters, to merge some of them with a higher similarity more than a threshold Thr_0;
(4) Output clustering result with K clusters until a termination condition is reached.

The below Algorithm 1 presents a brief implementation of Mashup services clustering.

3.3 Web APIs Recommendation Based on CF Algorithm for Mashup Cluster

As described in Fig. 2, when a user submits a Mashup development requirement, its textual description of the requirement is first converted to a Mashup requirement topic feature vector by using the LDA topic model. Probability of a word in a document can be assessed by the LDA based on probability of a topic in a document and probability of a word in the topic. The textual description of the requirement may be viewed as a mixture of various topics, and it can be characterized by a particular set of topics. Similarly, the same LDA-based process can be applied to the generated K Mashup service clusters, and the topic feature vector of each Mashup service cluster will be derived. Then, we use the JS divergence to measure the topic matching probabilities of all Mashup service clusters regarding the given Mashup requirement. Therefore, a Mashup service cluster with the highest matching similarity measurement with the Mashup requirement should be the most one satisfying the users' Mashup requirement in all clusters. Afterwards, a recommendation method is designed to rank and recommend top-R Web APIs within the Mashup service cluster with the highest similarity measurement. Since the number of Web APIs is huge while number of Mashup service clusters is small, we choose item-based CF algorithm as our recommendation method. It uses historical invocation records between Mashups clusters and Web APIs to rank and recommend diverse Web APIs, including popular and unpopular ones.

3.3.1 Problem Definition

Suppose that there are K Mashup service clusters $\{M_1, M_2, \ldots, M_K\}$ and N Web APIs $\{WA_1, WA_2, \ldots, WA_N\}$, the invocation relationship between Mashup service clusters and Web APIs can be represented by a matrix $R = [r_{ij}]_{K \times N}$, and the value of r_{ij} is equal to a normalized popularity of WA_j in M_i, $0 \leq r_{ij} \leq 1$, representing the total frequency of all Mashup services in M_i which historically invoked WA_j. If all

TABLE 2
The Invocation Relationship Matrix R

	WA_1	WA_2	WA_3	WA_4	WA_5
M_1	0.6	0.3	0.4	0.3	0
M_2	0	0.5	0.7	0.4	0
M_3	0.1	0.2	0.7	0	0.8

the Mashup services in M_i did not invoke WA_j, then $r_{ij} = 0$. As shown in Table 2, there are three Mashup clusters (i.e., M_1, M_2, M_3) and five Web APIs (i.e., WA_1, WA_2, WA_3, WA_4, WA_5). As M_3 does not invoke WA_4, M_3 is regarded as an active user and WA_4 is looked as a target item.

Algorithm 1. Mashup Service Clustering

Input: $\{MS = MS_1, \{MS_2, \ldots, MS_N, \{MS_i(Z_1, Z_2, \ldots, Z_T), Thr_0$ // *MS is the original Mashup services set for clustering;* $\{MS_i(Z_1, Z_2, \ldots, Z_T)$ *is the topic vector of* MS_i*, which is derived from the two-level topic model in Section 3.2.1 and random wank processes in Section 3.2.2* //

Output: *K Mashup Service Clusters* $\{M_1, M_2, \ldots, M_K\}$

For $i = 1$ ***to*** N ***For*** $j = 1$ ***to*** N ***For*** $t = 1$ ***to*** T
Calculate $D_{js}(MS_i, \{MS_j)$ *by the formula (7);*
$Sim(MS_i, MS_j) = D_{js}(MS_i, MS_j)$;
End For ***End For*** ***End For***
Construct *MSim by matrix element sim(*MS_i*,* MS_j*);*
$Avesim\ (MS_i) = \sum_{j=1}^{N} sim(MS_i, MS_j)\ /N$;
ASS = {Avesim(MS_1*), Avesim(*MS_2*), …, Avesim(*MS_N*)};*
For $K = 1$ ***to*** N' //*build* N' *atom-clusters by K-Means algorithm*//
Select MS_i *with the max average similarity in ASS;*
Divide MS_i *into* M_k*, and* ***Remove*** MS_i *from ASS;*
For $j = 1$ ***to*** N
IF*(Sim(*MS_j*,* MS_i*) > AveSim(*MS_i*))*
Divide MS_j *into Cluster* M_k*, and* ***Remove*** MS_j *from ASS;*
End For ***End For***
For $x = 1$ ***to*** N' ***For*** $y = 1$ ***to*** N'
// *hierarchical clustering for* N' *atom-clusters by Agnes algorithm*//
For $p = 1$ ***to*** P ***For*** $q = 1$ ***to*** Q
//*P, Q represent Mashup service's number in* M_x*,* M_y *respectively*//
$Sim\ (M_x, M_y) = \sum_{p=1}^{P} \sum_{q=1}^{Q} sim(MS_p, MS_q)/P * Q$;
Construct *CSim by matrix element sim(*M_x*,* M_y*);*
End For ***End For***
End For ***End For***
Do {Find M_x *and* M_y *with the highest similarity in CSim;*
If *(Sim(*M_x*,* M_y*) >* Thr_0*)*
Merge M_x *and* M_y *into a new Mashup Cluster;*
End If
Update *CSim, the similarity between the merged new cluster and other clusters is equal to the mean similarity of them;*
} ***While*** *(! (all clusters are merged into a cluster || Sim (*C_i*,*C_j*) <* Thr_0*))*
Return *K Mashup Service Clusters.*

3.3.2 Personalized Similarity between Web APIs

Person Correlation Coefficient (PCC) was used to calculate similarity in recommendation system. Item-based CF algorithm adopts PCC to calculate the similarity between WA_i and WA_j, which can be denoted as below formula:

$$sim\ (WA_i, WA_j) = \frac{\sum_{m \in M}\left(r_{m,WA_i} - \overline{r_{WA_i}}\right)\left(r_{m,WA_j} - \overline{r_{WA_j}}\right)}{\sqrt{\sum_{m \in M}\left(r_{m,WA_i} - \overline{r_{WA_i}}\right)^2}\sqrt{\sum_{m \in M}\left(r_{m,WA_j} - \overline{r_{WA_j}}\right)^2}} \tag{8}$$

Here, M is a set of Mashup service clusters that invoked both WA_i and WA_j, r_{m,WA_i} and r_{m,WA_j} represent the frequency of Mashup services in the cluster m historical invoked WA_i and WA_j respectively. $\overline{r_{WA_i}}$ and $\overline{r_{WA_j}}$ represent average frequency values of WA_i and WA_j invoked by different Mashup service clusters respectively. As for Table 2, the similarities between WA_4 and all other Web APIs are computed by using the formula (8), i.e., $sim(WA_4, WA_1) = 0.1$, $sim(WA_4, WA_2) = 0.83$, $sim(WA_4, WA_3) = 0.95$, $sim(WA_4, WA_1) = 0.1$.

3.3.3 Similar Neighbor Selection and Missing Value Prediction

After calculating the similarities between Web APIs, we obtain a Web APIs similarity matrix and apply top-K algorithms which widely used in recommendation system to identify a set of similar neighbors for WA_i, denoted as S(WA_i). Then we use S(WA_i) to predict the missing values for the target Web API by employing below formula:

$$P\left(r_{m,\,WA_i}\right) = \overline{WA_i} + \frac{\sum_{WA_{i_k} \in S(WA_i)} sim\left(WA_{i_k}, WA_i\right)\left(r_{m,\,WA_{i_k}} - \overline{WA_{i_k}}\right)}{\sum_{WA_{i_k} \in S(WA_i)} sim\left(WA_{i_k}, WA_i\right)} \tag{9}$$

Here, $\overline{WA_i}$ is an average frequency value of the target Web APIs item WA_i invoked by different Mashup service clusters, and $\overline{WA_{i_k}}$ is an average frequency value of the similar Web APIs item WA_{i_k} invoked by different Mashup service clusters. As we know from Section 3.3.2, the similarities between WA_4 and WA_3, WA_2 is large, we first select them as similar neighbors of WA_4, i.e., S(WA_4) = {WA_3, WA_2}, and then perform the missing value prediction of WA_4 to M_3 by using formula (9), i.e., $P_{M_3,WA_4} = 0.266$.

3.3.4 Web APIs Recommendation for Mashup Clusters

The predicted missing values can be employed to recommend optimal Web APIs for Mashup clusters. The recommendation list is generated by combining the known popularity values and the predicted missing values of Web APIs for a special Mashup service cluster. When an active user submits a Mashup requirement, the below process will be performed to recommend top-R Web APIs.

(1) Users' Mashup requirement is converted a topic feature vector T_{user} by applying LDA technology.
(2) For each Mashup cluster, we also obtain its topic feature vector T_{Mi} by appling LDA technology.
(3) Similarity matching $sim(T_{user}, T_{Mi})$ will be performed between T_{user} and T_{Mi} by JS divergence, a specific

Mashup cluster with the highest similarity will be assigned.

(4) For the specific Mashup cluster, a recommendation list with top-R Web APIs will be returned by implementing item-based CF algorithm.

Algorithm 2 presents the implementation of Web APIs recommendation for Mashup clusters.

Algorithm 2. Web APIs Recommendation

Input: *MR*, $M = \{M_1\, M_2, \ldots, M_K\}$, $WA = \{WA_1\, WA_2, \ldots, WA_N\}$ *// MR is users' Mashup development requirement description; M is a set of Mashup service clusters obtained in the Algorithm 1; WA is a set of Web APIs waiting for recommendation //*

Output: *Top-R Web APIs for Mashup cluster*

1. $T_{user} = LDA(MR)$;
2. ***For*** $i = 1$ *to* K
3. $T_{Mi} = LDA(M_i)$;
4. ***Calculate*** $sim(T_{user}, T_{Mi})$ *by formula (7);*
5. ***End For***
6. ***Rank*** $sim(T_{user}, T_{Mi})$ *and* ***Identify*** M_p *with the highest similarity;*
7. ***For*** $j = 1$ *to* P *// the number of Web APIs in Cluster* M_p *is P//*
8. ***For*** $k = j+1$ *to* P
9. ***Calculate*** $sim(WA_j, WA_k)$ *by formula (8);*
10. ***End For***
11. ***Select*** *top similar neighbors for* WA_j *and* ***construct*** $S(WA_j)$;
12. ***If*** *the value of* M_p *to* WA_j *is missing* ***then***
13. ***Predict*** *the missing values for* M_p *to* WA_j *by formula (9);*
14. ***End If***
15. ***End For***
16. ***For*** $l = 1$ *to* P *// the number of Web APIs in Cluster* M_p *is P //*
17. ***Rank*** *the popularity values or prediction values of* WA_l;
18. ***End For***
19. ***Return*** *top-R Web APIs for* M_p.

4 Experiments

To demonstrate the performance of our proposed method, the experiments on the Mashup service clustering and Web APIs recommendation respectively are conducted.

4.1 Experiment Dataset, Platform and Settings

We crawled 6,960 real Mashup services and their related data from the ProgrammableWeb site, as to November 2015. For each Mashup service, we first obtained the metadata of Mashup service's *category, name, descriptive text, Web APIs and tags.* After implementing the preprocessing on the crawled dataset, we obtained the standard content information of Mashup services. From the dataset, we know that the total amount of categories for 6,960 Mashup services is 445, and average number of Mashup services per category is 15.64. It has been also observed that the difference between numbers of Mashup services in categories is large. For example, category *Mapping* contains 1,038 Mashup services, but category *Addresses* contains only 1 Mashup service. Since small numbers of Mashup services lead to poor service clustering, we choose top 20 categories which involve 3,929 Mashup services and 62,078 words in the dataset as the experimental dataset. The detailed distribution information of Mashup services in top 20 categories is shown in Table 3, in which each category contains more than 50 Mashup services. The crawled dataset can be accessed and downloaded in the URL address http://49.123.0.60:8080/MashupNetwork2.0/dataset.jsp.

TABLE 3
The Distribution of Mashup Services in Top 20 Categories

Category	Number of Mashup Service	Category	Number of Mashup Service
Mapping	*1,038*	*Sports*	*112*
Search	*305*	*Telephony*	*99*
Social	*298*	*Blogging*	*98*
Ecommerce	*295*	*Reference*	*98*
Photos	*260*	*Electronic Signature*	*95*
Music	*251*	*Widgets*	*86*
Travel	*192*	*Visualizations*	*78*
Video	*174*	*Humor*	*67*
Messaging	*137*	*Government*	*66*
Mobile	*126*	*Games*	*54*

Based on the dataset and relationships among Mashup services, we have developed a Mashup Service Network Platform (http://49.123.0.60:8080/MashupNetwork2.0). In our experiment, we used the platform to construct the *MSN*, and employed random walk process on the *MSN*, which served as a basis of service clustering. For the simplicity of experiment, we set λ_1 and λ_2 in formula (1) as 1/2. Besides, according to the existing experience [23], [24], we set γ in formula (5) to 1. When using two-level topic model to perform unsupervised learning, Dirichlet hyperparameters α is set to 0.01, and Gibbs sampling iteration is set to 2,000. In Algorithm 1, we choose the best value of Thr_0 to produce optimal Mashup clustering quality.

4.2 Mashup Service Clustering

To consolidate the relationship among Mashups on clustering accuracy, we perform service clustering experiments, which is as a basis of Web APIs recommendation.

4.2.1 Evaluation Metrics

We choose precision and recall from information retrieval to evaluate the accuracy of Mashup service clustering. Actually, Mashup service clustering is a classification process. The precision for a class is the number of true positives (TP) (i.e., the number of items correctly labeled as belonging to the class) divided by the total number of elements labeled as belonging to class (i.e., the sum of TP and false positives (FP), which are items incorrectly labeled as belonging to the class). Recall for a class is the number of TP divided by the total number of elements that actually belong to the class (i.e., the sum of TP and false negatives (FN), which are items which were not labeled as belonging to the class but should have been). We firlstly created a document to include a standard/positive classification result for all original Mashup service documents *MS*, which can be jointly determined by domain experts and users. Then, we represent the standard/positive classification result for *MS* as $SM = \{SM_1, SM_2, \ldots, SM_V\}$, and the experimental

service clustering result as $M = \{M_1, M_2, \ldots, M_K\}$. So the precision and recall are respectively defined as formulas (10) and (11) below.

$$Precision\ (M_i) = \frac{TP}{TP + FP} = \frac{|SM_i \cap M_i|}{|M_i|} \tag{10}$$

$$Recall\ (M_i) = \frac{TP}{TP + FN} = \frac{|SM_i \cap M_i|}{|SM_i|} \tag{11}$$

Here,

- TP—the number of Mashup services in Mashup cluster M_i correctly labeled as belonging to the corresponding Mashup cluster SM_i;
- FP—the number of Mashup services in M_i incorrectly labeled as belonging to SM_i;
- FN—the number of Mashup services which were not labeled as belonging to SM_i but should have been;
- $|SM_i|$—the number of Mashup services in SM_i;
- $|M_i|$—the number of Mashup services in M_i;
- $|SM_i \cap M_i|$—the number of Mashup services jointly appeared in SM_i and M_i.

Besides, we also employ purity and entropy to evaluate the accuracy of service clustering. The purity of each M_i and the mean purity of all Mashup service clusters in M are respectively defined in the formulas (12) and (13). Similarily, the entropy of each M_i and the mean entropy of all Mashup service clusters in M are respectively defined in the formulas (14) and (15).

$$Purity\ (M_i) = \frac{1}{|M_i|} \max_j n_i^j,\ 1 \le i \le K,\ 1 \le j \le V \tag{12}$$

$$Purity\ (M) = \sum_{i=1}^{K} \frac{|M_i|}{|MS|}\ Purity(M_i) \tag{13}$$

$$Entropy\ (M_i) = -\frac{1}{logV} \sum_{j=1}^{V} \frac{n_i^j}{|M_i|} log\left(\frac{n_i^j}{|M_i|}\right) \tag{14}$$

$$Entropy\ (M) = \sum_{i=1}^{K} \frac{|M_i|}{|MS|}\ Entropy(M_i) \tag{15}$$

Here, $|M_i|$ is the number of Mashup services in cluster M_i, n_i^j is the number of Mashup services belong to SM_j which are successfully divided into M_i, and $|MS|$ is the total amount of Mashup services in the MS. In short, the bigger recall, precision, purity and the smaller entropy, mean that the clustering accuracy is the better.

4.2.2 Baseline Methods

We compare our method with three existing methods:

- *K-Means*. Both description text and tags of Mashup services are used. K-Means is used to cluster Mashup services according to the composite similarity [25].
- *LDA*. *LDA* technique is used to cluster Mashup services, in terms of both their description text and tags [20]. The similarities between services are measured by KL divergence of services' topic feature vectors [1].

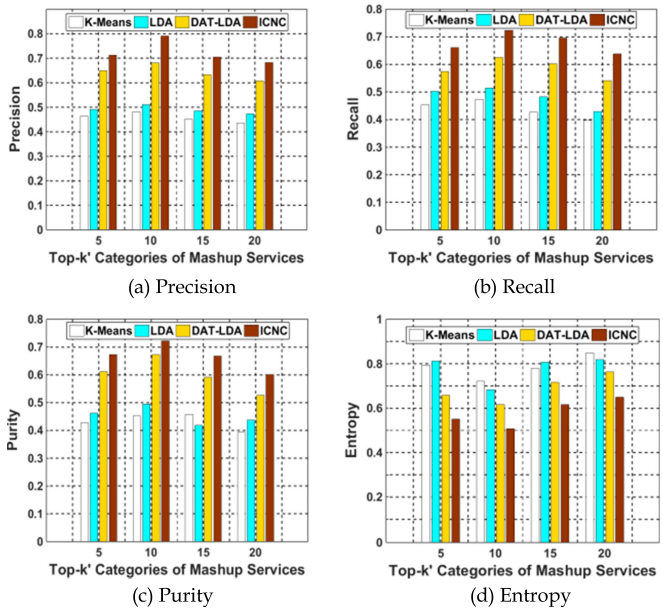

Fig. 5. Mashup services clustering performance comparison.

- *DAT-LDA*. In our previous work [22], we proposed a Mashup service clustering method by using LDA model built from multiple data sources.
- *ICNC*. The proposed clustering method in this paper, which integrates Mashup service's content (description, Web APIs and tags) and network by using a two-level topic model to cluster Mashup service.

4.2.3 Experimental Results

We compare different approaches to evaluate their clustering performance in terms of precision, recall, purity and entropy, and investigate the impact of the parameter T and β on the clustering result in our *ICNC* approach.

(1) Clustering Performance Comparison. To study the clustering performance, we compare our method with other three baseline methods. As for *ICNC*, we choose optimal T and β for different sets of categories to achieve the best performance. A detailed investigation of T and β will be described in subsequent Section. Fig. 5 reports the comparisons on the ProgrammableWeb dataset when the number of categories (clusters) in Table 3 ranging from 5 to 20 with a step 5 (i.e., top-k' = 5/10/15/20). The comparisons show that our *ICNC* significantly improves the clustering accuracy and outperforms all other baseline methods in terms of the mean values of precision, recall, purity and entropy.

Specifically, we have the below observations:

- The performance of *K-Means* is the worst in most cases. This is because *K-Means* only uses the term-based vector space model to represent the functional features of Mashup services without considering latent semantic correlation behind the terms of Mashup service documents. *LDA*, *DAT-LDA* and *ICNC* all exploit the LDA technique to mine latent functional topics from Mashup service documents to improve clustering accuracy. Compared with *LDA*, *DAT-LDA* shows the better performance since it simultaneously takes multiple data sources and a hybrid clustering algorithm

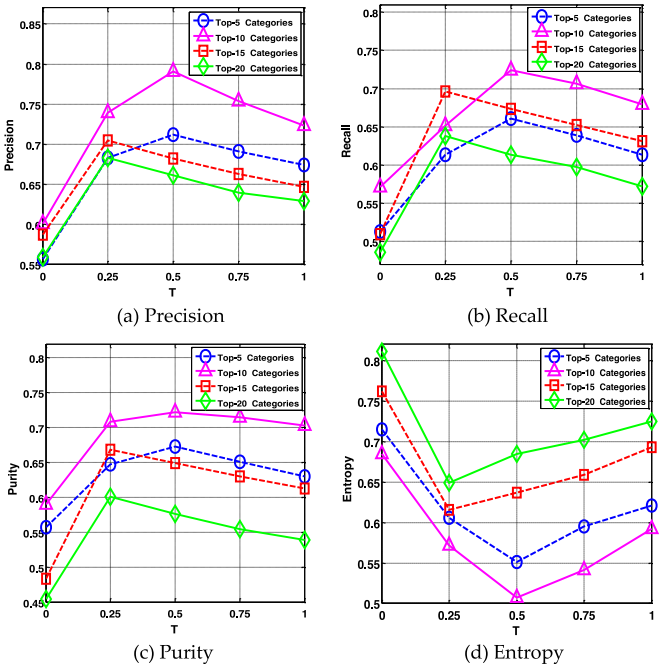

Fig. 6. Impact of T on mashup service clustering results.

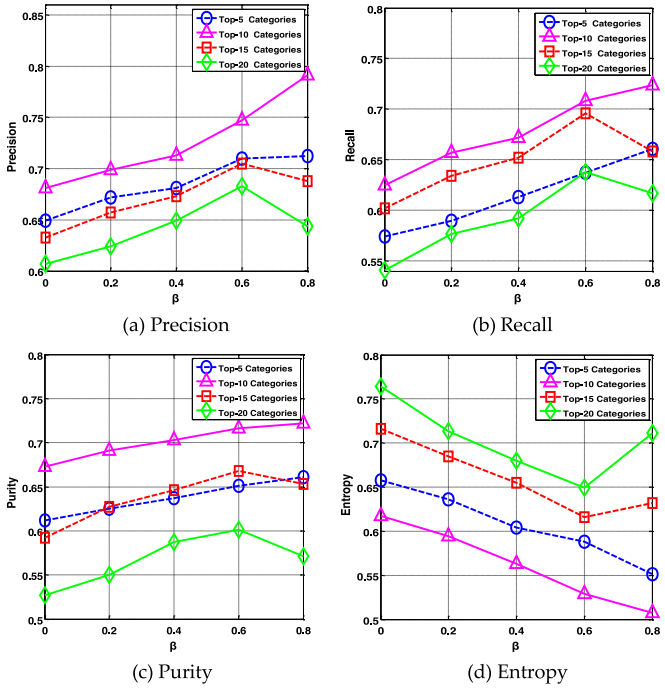

Fig. 7. Impact of β on mashup service clustering results.

into consideration. The two-level topic model of *ICNC* considers relationships among Mashup services and derives useful topics from Mashup service network, which significantly improves clustering accuracy.

- The performance of all methods in top-10 categories surpasses to those of top-5, 15 and 20 categories in most cases. When the number of categories is small, an increase in the number of categories (e.g., from 5 to 10) improves clustering performance, since more Mashup services in these categories can be used to construct MSN for better clustering accuracy. However, when the number of categories continues to increase from 10 to 20, the clustering accuracy decreases. The reason for this is that additional categories only have fewer Mashup services with less linked relationships, which leads to a few isolated nodes in the MSN and therefore weakens the clustering accuracy. The observations indicate that it is important to choose an appropriate number of categories for clustering.

(2) Impacts of T and β. The experiments investigate the impact of parameter T on Mashup service clustering in our *ICNC*. During the experiments, we select the best values of β for different sets of categories (*i.e.*, $\beta = 0.8$ *for top 5 and 10 categories,* $\beta = 0.6$ *for top 15 and 20 categories).* We change the value of T from *0* to *1* with a step of *0.25*, and obtain the mean values of precision, recall, purity and entropy in Fig. 6. The experimental results indicate that when we choose $T = 0.5$ for top 5 and 10 categories and $T = 0.25$ for top 15 and 20 categories, the precision, recall, purity and entropy of them reach their peak values. When T increases from 0.5 to 1, the clustering performances at all different sets of categories constantly decrease. The reason for this is a larger T leads to smaller *MSN* with high-similarity Mashup services, in which the number of incorporated Mashup services from network level to content level becomes smaller. In addition, when T decreases from 0.25 to 0, the clustering performance at all different sets of categories also constantly decrease. Smaller T means a larger *MSN* with more low-similarity Mashup services, which may bring too much noise and decrease clustering performance.

Fig. 7 shows the impact of different β from *0* to *0.8* with a step of 0.2 on Mashup service clustering in our *ICNC*. Similarly, we select the best values of T for different sets of categories *(i.e.,* $T = 0.5$ *for top 5 and 10 categories,* $T = 0.25$ *for top 15 and 20 categories).* The experimental results indicate that when we choose $\beta = 0.8$ for top 5 and 10 categories and $\beta = 0.6$ for top 15 and 20 categories, the mean values of precision, recall, purity and entropy reach their peak values. As expected, when β closes to 0, the performance of our *ICNC* similar to *DAT-LDA,* since the relationships among Mashup services are not considered at all. As β becomes larger, *ICNC* provides a better clustering performance in most cases of different sets of categories, which indicates that a larger β achieves a better clustering performance when the linked Mashup services in the *MSN* are closely related to each other.

4.3 Web APIs Recommendation

To solidify the implicit co-invocation relationship between Web APIs on top of Mashup service clustering results in Section 4.2, on recommendation accuracy and diversity, we perform Web APIs recommendation experiments.

4.3.1 Evaluation Metrics

We evaluate the performance of different recommendation methods in terms of accuracy and diversity from information retrieval. Discounted Cumulative Gain@top R (*DCG@R*) is a measure of ranking quality, where R is the number of Web APIs in the recommendation list. It is often used to measure the accuracy of recommendation based on the relevance of top-R recommended items in the recommendation list. The bigger *DCG@R*, the better recommendation accuracy. It is defined as below:

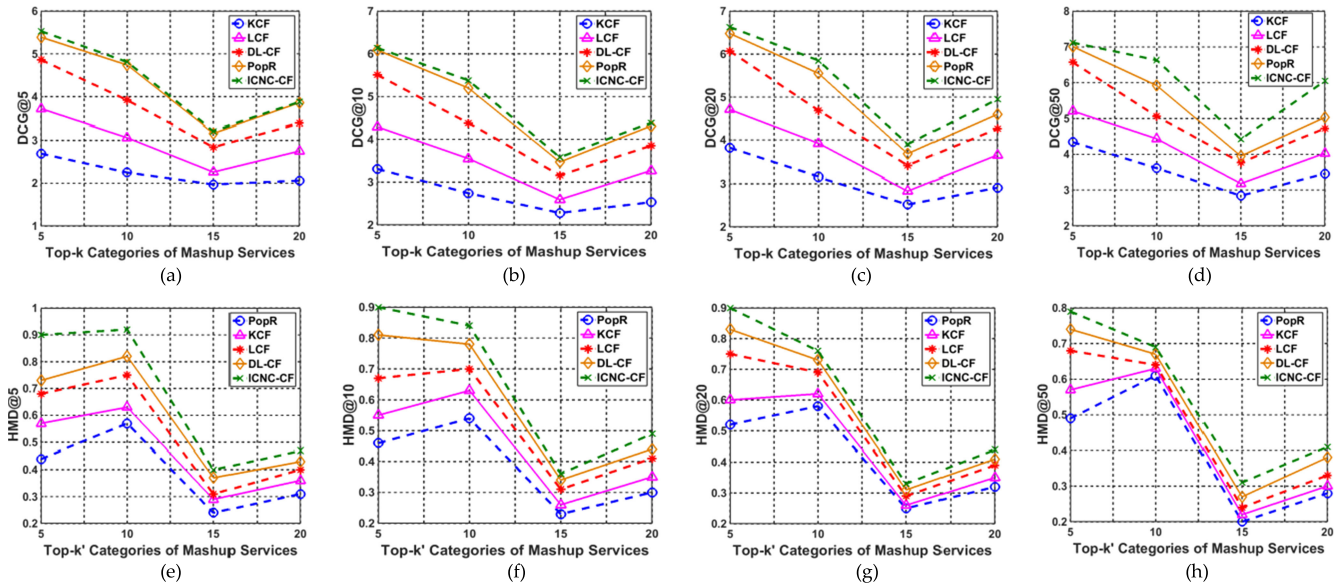

Fig. 8. Web APIs recommendation performance comparison.

$$DCG@R = \sum_{i=1}^{R} \frac{2^{r(i)} - 1}{log_2(1+i)} \quad (16)$$

Here, i is the rank position in the top-R Web APIs recommendation list, $r(i)$ is the relevant score of the ith rank position on the top-R recommendation list, $0 \leq r(i) \leq 1$. If the recommended Web API is used by the Mashup cluster, then $r(i)$ is equal to the normalized popularity of the Web API, otherwise the normalized predicted missing value of Web API obtained by formula (9) in Section 3.3.3.

Moreover, we use Hamming Distance (*HMD*) to evaluate the diversity of recommendation. Hamming distance measures the quality of recommending diverse Web APIs for different Mashup requirement. A higher hamming distance means better diversity. It is defined as below:

$$HMD(m_i, m_j)@R = 1 - \frac{Q(m_i, m_j)}{R} \quad (17)$$

Here, $Q(m_i, m_j)$ represents the number of common Web APIs in the top-R recommended lists of Mashup service clusters m_i and m_j, R is the total length (the number of Web APIs) in the recommended list. If the Web APIs recommendation lists of m_i and m_j are common, $H(m_i, m_j) = 0$; If there are not any common Web APIs in their recommendation lists, $H(m_i, m_j) = 1$.

4.3.2 Baseline Methods

We compare our method with four baseline methods on top of multiple clustering methods in Section 4.2.2.

- *PopR*. This method recommends Web APIs which are most popular in the matching Mashup cluster for user's Mashup requirement. The popularity of a Web API is measured by the number of Mashup service in the cluster which contains the Web API. It ranks and recommends Web APIs with top-R popularity.
- *KCF*. This method first uses K-Means algorithm to cluster Mashup service into different Mashup clusters (categories) according to the composite similarity [25], then applies item-based CF algorithm to rank and recommend top-R Web APIs for Mashup clusters. The method has been adopted in research work [18].
- *LCF*. In this method, LDA is used to cluster Mashups according to services' topic feature vectors [1], [20]. Item-based CF algorithm is employed to rank and recommend top-R Web APIs for Mashup cluster.
- *DL-CF*. In this method, DAT-LDA is applied to cluster Mashups based on services' description, Web APIs and tags [22]. Item-based CF algorithm is used to rank and recommend top-R Web APIs for Mashup cluster.
- *ICNC-CF*. The proposed method in this paper. It considers the relationship among Mashup services, and integrates Mashup service's content (description, Web APIs and tags) and network by using a two-level topic model to cluster Mashup service. It also considers the historical invocation history between Mashups clusters and Web APIs, and exploits item-based CF algorithm to rank and recommend diverse Web APIs (popluar and unpopular) for users' Mashup requirement.

4.3.3 Experimental Results

Fig. 8 reports the comparisons of experimental results when the number of categories in Table 3 ranging from 5 to 20 with a step 5 (i.e., top-k' = 5/10/15/20). The comparisons show that our *ICNC-CF* achieves a better recommendation performance over all other baseline methods in all cases in terms of the mean values of *DCG* and *HMD* with different number of recommended Web APIs (i.e., R = 5/10/20/50). Specifically, we have below observations:

- The recommendation accuracy (i.e., *DCG*) of *ICNC-CF* in Figs. 8a, 8b, 8c and 8d significantly outperforms *DL-CF*, *LCF* and *KCF* consistently, and slightly surpasses *PopR*. The clustering method in *ICNC-CF* considers the relationships among Mashup services

and exploits more useful topics for clustering, which significantly improves recommendation accuracy. Moreover, same clustering approach used in ICNC-CF and *PopR*, making the difference of recommendance accuracy small. The slight advantage of *ICNC-CF* to *PopR* illustrates the popularity remain plays a key role in the Web APIs recommendation. Even so, our *ICNC-CF* recommends popular and unpopular Web APIs for Mashup cluster by using CF algorithm.

- The recommendation diversity (i.e., *HMD*) of *ICNC-CF* in Figs. 8e, 8f, 8g and 8h significantly exceeds *DL-CF*, *LCF*, *KCF* and *PopR*. The *HMD* of *PopR* is the worst. This is because *PopR* only uses the popularity to rank and recommend Web APIs for Mashup service cluster without considering the implicit co-invocation relationship between Web APIs, resulting in single recommendation lists with many common Web APIs. *ICNC-CF*, *DL-CF*, *LCF* and *KCF* all exploit CF technique to mine the implicit co-invocation relationship between Web APIs to improve the diversity of recommendation. Compared with *DL-CF*, *LCF* and *KCF*, *ICNC-CF* shows better performance since it simultaneously considers the accuracy and diversity of recommendation.
- The performance of all methods in top-15 categories in Fig. 8 are the worst, i.e., the lowest points of the DCG and HMD in all methods appear when $k' = 15$. The reason for this is that the total number of common historical Web APIs invocation records for different Mashup services in top-15 categories decreases. That is to say, compared to top-5, 10 and 20, the addition of Mashup categories "Sports", "Telephony", "Blogging", "Reference" and "Electronic Signature" in Table 3 lowers the total number of common historical Web APIs invocation records, and thus decreases the accuracy of personalized similarity between Web APIs, missing value prediction of the target Web API and DCG measurement. Besides, the more sparse Web APIs invocation records, the more common Web APIs appeared in the top-R recommendation lists of Mashup service clusters. This directly leads to the decreasing of DCG.

5 Related Work

The related works are investigated in three parts: service clustering, service recommendation and clustering-based service recommenation.

5.1 Service Clustering

Web service clustering technology plays an important role in service searching and effectively improves the quality of service discovery and recommendation [1], [26]. A number of research works have been done on it.

Service documents are a main information sources for service clustering [27]. The functional feature vectors of service generally are characterized as a term-based vector space model by processing service document [19]. The similarity among services was measured by using similarity methods, such as cosine similarity. In addition, tags were considered as an important data source to boost performance of service clustering. Especially, L. Chen et al. used both WSDL documents and tags to cluster web services, and proposed WTCluster [25] and WT-LDA [20] respectively. However, only using a limited number of terms and tags for similarity measurement in these works, it may result in unsatisfactory clustering accuracy.

Several recent works show a promising advancement in clustering accuracy through mining latent functional factors from service documents [28]. Factor analysis [29], topic model [21] and matrix factorization [3] are used to identify the latent functional factors and discover implicit semantic correlation among service documents. Where, Yu et al. identified suitable number of latent functional factors for large-scale service clustering [28]. These methods definitely boost service clustering performance. However, few of them perceive service documents are related to each other in many ways instead of being independent [30] and none of them uses relationship for service clustering. The relationship among services can be used to derive useful topics for achieving better clustering accuracy. Guo et al. [30] proposed a two-level topic model towards knowledge discovery from citation networks to discover latent topics and recommend important citations. Tang et al. [23], [24] developed a topic modeling and its integration into random walking framework for academic search.

Motivated by above approaches and our observation of impact of relationships among Mashup services on their clustering, we develop a two-level topic model to mine more useful topics for better service clustering.

5.2 Service Recommendation

Many research works have been performed in service recommendation, which can be reviewed as four categories: content-based [2], structure-based [11], [12], [31], QoS-based [7], [8], [9], [10], and hybrid methods [14], [15], [16], [17].

First of all, content-based service recommendation mainly explores similarity matching between users' requirement and services to rank and recommend services with high-matching [2], [6]. The similarity commonly is measured by using services' functionality description. Liu et al. [2] proposed to use collaborative topic regression to recommend Web APIs.

Second, recommending Web APIs needs to identify components that can fit the Mashup under composition as well as from the structural perspective—not only from the content point of view. Especially, Greenshpan et al. presented an efficient and intuitive development technique of Mashup based on a novel autocompletion mechanism [11]. It uses similarities between the ways users glue together Mashup components and recommends possible completions for a user's partial Mashup specification, based on the syntactic inheritance of Mashlet components and GP compositions. The inheritance plays a central role in autocompletion mechanism, a Mashlet is modeled as its set of input and output relations, and a GP is modeled as a graph of connections between the input and output relations of the mashlets that it links. However, the inheritance relationship among Mashlet/GP is currently not recorded in the ProgrammableWeb repository. Different from this work, the way Web APIs are glues together in our paper is clearly specified by existing

Web APIs composition in common Mashups or derived by the historical invocations between Mashups and Web APIs. Soliman et al. proposed a Mashup authoring and processing system MashRank, based on concepts from rank-aware processing, probabilistic databased, and information extraction to enable ranked Mashups of unstructured sources with uncertain ranking attributes [12].

Third, Quality of Service (QoS) plays a very important role in Web services recommendation [7], [8], [9], [10]. Zheng et al. [7] presented a QoS-aware Web service recommendation WSRec. Fletch et al. [8] proposed an elastic personalized nonfunctional attribute preference and trade-off based service recommendation. Wang et al. [9] explored reputation measurement and malicious feedback rating prevention in Web service recommendation system. More importantly, the quality of Mashup under construction needs to be considered. The recommended APIs should somehow help the developers improve their Mashup applications. Cappiello et al. first defined a Mashup component quality model from the perspectives of API quality, data quality and presentation quality and identified a set of related quality dimensions and metrics [35]. Then they designed a quality model for Mashups, which emphasizes the component-based nature of Mashups and focuses on composition quality such as added value, component suitability, component usage, consistency, and Mashup availability [36]. They also used the quality as a driving factor for recommendation and developed a tool for Mashup design [37], [38].

Hybrid service recommendation integrated the above methods to recommend services. Kang et al. [14] incorporated functional interest, QoS preference, and diversity feature to recommend top-k diversified Web services. Zhong et al. [15] explored service evolution, collaborative filtering and content matching for time-aware service recommendation. Li et al. [16] used a relational topic model to recommend Web API in Mashup creation. Yao et al. [17] first proposed a novel hybrid Web service recommendation method by using a three-way aspect model, and then explored both explicit textual similarity and implicit correlation of APIs to make API recommendation [5].

5.3 Clustering-Based Service Recommendation

Although clustering and recommending are two well-known techniques in service-oriented computing, they are usually regarded as two independent processes [32], which may result in poor and single recommendation results. Currently, few researchers addressed the problem and incorporated service clustering into WSDL/SOAP based Web service recommendation [32], [33], [34]. Most recently, Gao et al. [18] categorized existing Mashup into functionally similar clusters, and then recommended Web APIs for each Mashup cluster using a manifold ranking algorithm. The work used TF-IDF to measure the similarity between service documents without considering latent semantic correlation behind the terms of service documents. Xia et al. [1] proposed a category-aware API clustering and distributed recommendation method. They improved the similarity measurement in service clustering to provide a basis for Web APIs recommendation by using a extend K-Means algorithm based on LDA. This method only used independent Mashup services for clustering without considering relationship among them. Yao et al. [5] investigated the historical invocation relations between Web APIs and Mashups to infer the implicit correlations among Web APIs, and incorporated the correlations into matrix factorization model for service recommendation. The diversity of recommendation is not addressed in this work. In fact, Mashups with similar functionality can be grouped as a cluster, the invocation history between Mashups and Web APIs can be expanded to Mashup cluster and Web APIs [18]. Consenquently, the sparsity will be largely relieved, and more implicit correlation among Web APIs will be mined to recommend diverse Web APIs.

6 Discussion

In our paper, the ways Web APIs are combined together include: (1) the existing Web APIs composition in common Mashups; (2) the implicit correlation of Web APIs derived by the historical invocations between Mashup clusters and Web APIs. Some useful topics are derived from the constructed service network to calculate the similarity between Mashup services. Mashup services with similar functionalities are clustered. A list of similar Web APIs in Mashup service clusters is recommended for Mashup development. The accuracy of Web APIs recommendation is significantly improved as a result. On the other hand, the implicit correlation of Web APIs is exploited to rank and recommend a list of diverse Web APIs for Mashup clusters. The total frequency of Web APIs within Mashup clusters is used to identify a set of similar neighbors, and predict the missing values of Web APIs. The frequency or prediction values of Web APIs are ranked. A diverse Web APIs set is recommended for the given Mashup cluster. The diversity of Web APIs recommendation is also achieved.

Cappiello et al. [39] analyzed most popular Mashups published on ProgrammableWeb and identified the master and slave roles of components in the Mashup under construction. The master component is the one user interacts with the most. The slave component's behavior depends on the master component. As mentioned in the Section 1, the top 200 most popular Web API invocations in Mashups cover about 99 percent of all Web APIs invocations in Mashups. Only a few popular Web APIs are invoked by many Mashups. The popular Web APIs can be considered as master components, while the unpopular even unused Web APIs can be considered slave components. When Web APIs are selected for Mashup development, the master components with higher popularity are preferred from all matching Web APIs components with similar functionality. The slave components are ranked and recommended by integrating their functional similarity and popularities or prediction values so that they can work with the master component for providing a composable Web APIs set.

The compatibility of recommended Web APIs with respect to the Web APIs already in Mashups should be considerd. It represents the composition possibility of one recommended Web API with those already included in a Mashup. Different from SOAP/WSDL-based Web services, RESTful Web API do not have a standard description of service capability, making Mashup harder compared with traditional Web service composition that uses syntactic and semantic matching between input/output parameters.

Mashup construction also is not trivial as the names of Web APIs' input/output variables are not always meaningful or uniform [11]. Inconsistency/incom patibility between Web APIs may cause failure of Mashups. This type of composition is not an automatic process and needs manual intervention or assistance. User-driven inconsistency resolution technique may be a good solution [39].

More fine-granularity (for example, *API function-level*) based Mashup construction can be further explored. Additional service composition relationship can be mined and used to build novel Mashup application based on users' personal requirement. However, due to the incompatibility/inconsistency between API functions (for example, *different protocols, languages and data formats, different data types between input/output parameters exposed by APIs [37]*), the API function-level Mashup construction may not be accurate and even fail. In this paper, only Web API-level Mashup construction is considered. The selection of special API functions depends on users' personal requirement. It is suggested to identify "right" APIs by considering technological, syntactic, and semantic compatibility of each API within Mashup construction [38]. A type of API interface is proposed to specify how API functions interact with each other as well as composition rules [11].

Quality-driven Mashup development technique is attractive. Recently, several works [35], [36], [37], [38] have concentrated on the quality issue of Mashups and their components to help a Mashup composer or user in the selection and recommendation of components and composition. Additional in-depth researches on quality-driven Mashup development technique can be performed, such as further investigation of metrics of quality attributes and optimal Web APIs composition with quality constraints.

7 Conclusion and Future Work

This paper presents an integrated content and network-based service clustering and Web APIs recommendation method for Mashup development. A two-level topic model based on an integration of service content and network is developed to mine topics for more accurate service clustering. In the model, via two random walk processes, the novel topics are derived from the linked Mashup documents at the network level and incorporated into the topic probability distribution of original Mashup service documents at the content level. Moreover, a CF-based Web APIs recommendation algorithm is proposed to recommend diverse Web APIs for Mashup clusters, via inferring and using the historical invocation history between Mashups clusters and Web APIs. The comparative experiments performed on ProgrammableWeb dataset demonstrate the effectiveness of the proposed method and show that our method significantly improves the accuracy and diversity of Web APIs recommendation. In the future work, we will focus on one or more topics mentioned in Section 6, to facilitate and improve Mashup development.

Acknowledgments

The work was supported by the National Natural Science Foundation of China under grant No. 61572371, 61572186, 61572187, 61402167, 61402168, SKLSE of China (Wuhan University) under grant No mean. SKLSE2014-10-10.

References

[1] B. Xia, Y. Fan, W. Tan, K. Huang, J. Zhang, and C. Wu, "Category-aware API clustering and distributed recommendation for automatic mashup creation, *IEEE Trans. Services Comput.*, vol. 8, no. 5, pp. 674–687, Sep./Oct. 2015.

[2] X. Liu and I. Fulia, "Incorporating user, topic, and service related latent factors into web service recommendation," in *Proc. IEEE Int. Conf. Web Serv.*, 2015, pp. 185–192.

[3] Z. Zheng, H. Ma, M. Lyu, and I. King, "Collaborative web service QoS prediction via neighborhood integrated matrix factorization," *IEEE Trans. Serv. Comput.*, vol. 6, no. 3, pp. 289–299, Jul.-Sep. 2013.

[4] W. Xu, J. Cao, L. Hu, J. Wang, and M. Li, "A social-aware service recommendation approach for mashup creation," in *Proc. IEEE 20th Int. Conf. Web Serv.*, 2013, pp. 107–114.

[5] L. Yao, X. Wang, Q. Sheng, W. Ruan, and W. Zhang, "Service recommendation for mashup composition with implicit correlation regularization," in *Proc. IEEE 20th Int. Conf. Web Serv.*, 2015, pp. 217–224.

[6] L. Liu, F. Lecue, and N. Mehandjiev, "Semantic content-based recommendation of software services using context," *ACM Trans. Web*, vol. 7, no. 3, pp. 17–20, 2013.

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

[8] K. Fletcher, F. Liu, and M. Tang, "Elastic personalized nonfunctional attribute preference and trade-off based service selection," *ACM Trans. Web*, vol. 9, no. 1, pp. 1–26, 2015.

[9] S. Wang, Z. Zheng, Z. Wu, M. Lyu, and F. Yang, "Reputation measurement and malicious feedback rating prevention in web service recommendation system," *IEEE Trans. Serv. Comput.*, vol. 5, no. 8, pp. 755–767, Sep./Oct. 2015.

[10] M. Tang, Y. Jiang, J. Liu, and X. Liu, "Location-aware collaborative filtering for QoS-based service recommendation," in *Proc. IEEE 20th Int. Conf. Web Serv.*, 2012, pp. 202–209.

[11] O. Greenshpan, T. Milo, and N. Polyzotis, "Autocompletion for mashups," *Proc. VLDB Endowment*, vol. 2, pp. 538–549, 2009.

[12] M. A. Soliman, I. F. Iiyas, and M. Saleeb, "Building ranked mashups of unstructured sources with uncertain information," *Proc. VLDB Endowment*, vol. 3, pp. 826–837, 2010.

[13] B. Cao, et al., "Mashup service clustering based on an integration of service content and network via exploiting a two-level topic model," in *Proc. IEEE 20th Int. Conf. Web Serv.*, 2016, pp. 212–219.

[14] G. Kang, M. Tang, J. Liu, X. Liu, and B. Cao, "Diversifying Web Service recommendation results via exploring service usage history," *IEEE Trans. Serv. Comput.*, vol. 9, no. 4, pp. 566–579, Jul./Aug. 2016.

[15] Y. Zhong, Y. Fan, K. Huang, W. Tan, and J. Zhang, "Time-aware service recommendation for mashup creation in an evolving service ecosystem," in *Proc. IEEE 20th Int. Conf. Web Serv.*, 2014, pp. 25–32.

[16] C. Li, R. Zhang, J. Huai, and H. Sun, "A novel approach for API recommendation in mashup development," in *Proc. IEEE 20th Int. Conf. Web Serv.*, 2014, pp. 289–296.

[17] L. Yao, Q. Z. Sheng, H. H. Ngu, Y. Yu, and A. Segev,"Unified collaborative and content-based Web Service recommendation," *IEEE Trans. Serv. Comput.*, vol. 8, no. 3, pp. 453–466, May/Jun. 2015.

[18] W. Gao, L. Chen, J. Wu, and H. Gao, "Manifold-learning based API recommendation for mashup creation," in *Proc. IEEE 20th Int. Conf. Web Serv.*, 2015, pp. 432–439.

[19] B. Cao, J. Liu, Z. Zheng, and G. Wang, "Mashup service recommendation based on user interest and social network," in *Proc. IEEE 20th Int. Conf. Web Serv.*, 2013, pp. 99–106.

[20] L. Chen, Y. Wang, Q. Yu, Z. Zheng, and J. Wu, "WT-LDA: User tagging augmented LDA for Web Service clustering," in *Proc. Int. Conf. Serv.-Oriented Comput.*, 2013, pp. 162–176.

[21] D. Blei, A. Ng and M. Jordan, "Latent dirichlet allocation," *J. Mach. Learning Res.*, vol. 3, pp. 993–1022, 2003.

[22] B. Cao, X. Liu, J. Liu, and M. Tang, "Effective mashup service clustering method by exploiting LDA topic model from multiple data sources," in *Proc. 9th Asia-Pacific Serv. Comput. Conf. Adv. Serv. Comput.*, 2015, pp. 165–180.

[23] J. Tang, R. Jin, and J. Zhang, "A topic modeling approach and its integration into the random walk framework for academic search," in *Proc. 8th IEEE Int. Conf. Data Min.*, 2008, pp. 1055–1060.

[24] Z. Yang, J. Tang, J. Zhang, J. Li, and B. Gao, "Topic-level random walk through probabilistic model," in *Proc. Adv. Data Web Manage.*, 2009, pp. 162–173.

[25] L. Chen, L. Hu, J. Wu, and Z. Zheng, "WTCluster: Utilizing tags for web service clustering," in *Proc. 9th Int. Conf. Serv.-Oriented Comput.*, 2011, pp. 204–218.
[26] K. Elgazzar, A. Hassan, and P. Martin, "Clustering WSDL documents to bootstrap the discovery of Web Services," in *Proc. IEEE Int. Conf. Web Serv.*, 2010, pp. 147–154.
[27] L. Chen, Q. Yu, P. Yu, and J. Wu, "WS-HFS: A heterogeneous feature selection framework for Web Services mining," in *Proc. IEEE Int. Conf. Web Serv.*, 2015, pp. 193–200.
[28] Q. Yu, H Wang, and L Chen, "Learning sparse functional factors for large-scale service clustering," in *Proc. IEEE Int. Conf. Web Serv.*, 2015, pp. 201–208.
[29] C. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. New York, NY, USA: Springer-Verlag New York. Inc., 2006.
[30] Z. Guo, Z. Zhang, S. Zhu, Y. Chi, and Y. Gong, "A two-level topic model towards knowledge discovery from citation networks," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 780–794, Apr. 2014.
[31] K. Huang, Y. Fan, W. Tan, and X. Li, "Service recommendation in an evolving ecosystem: A link prediction approach," in *Proc. IEEE Int. Conf. Web Serv.*, 2013, pp. 507–514.
[32] Y. Zhou, et al., "Ranking services by service network structure and service attributes," in *Proc. IEEE Int. Conf. Web Serv.*, 2013, pp. 26–33.
[33] D. Skoutas, D. Sacharidis, and T. Sellis, "Ranking and clustering web services using multicriteria dominance relationships," *IEEE Trans. Serv. Comput.*, vol. 3, no. 3, pp. 163–177, Jul.-Sep. 2010.
[34] Y. Xu, J. Yin, and Y. Li, "A collaborative framework of web service recommendation with clustering-extended matrix factorisation," *Int. J. Web Grid Serv.*, vol. 12, no. 1, pp. 1–25, 2016.
[35] C. Cappiello, F. Daniel, and M. Matera, "A quality model for mashup components," in *Proc. Int. Conf. Web Eng.*, 2009, pp. 236–250.
[36] C. Cappiello, F. Daniel, A. Koschmider, M. Matera, and M. Picozzi, "A quality model for mashups," in *Proc. Int. Conf. Web Eng.*, 2011, pp. 137–151.
[37] M. Picozzi, M. Rodolfi, C. Cappiello, and M. Matera, "Quality-based recommendations for mashup composition," in *Proc. Int. Conf. Web Eng.*, 2010, pp. 360–371.
[38] C. Cappiello, M. Matera, M. Picozzi, F. Daniel, and A. Fernandez, "Quality-aware mashup composition: Issues, techniques and tools," in *Proc. 8th Int. Conf. Quality Inf. Commun. Technol.*, 2012, pp. 10–19.
[39] C. Cappiello, F. Daniel, M. Matera, and C. Pautasso, "Information quality in mashups," *IEEE Internet Comput.*, vol. 14, no. 4, pp. 14–22, Jul./Aug. 2010.

Buqing Cao is currently an associate professor in the School of Computer Science and Engineering, Hunan University of Science and Technology, China. His current interests include service computing and software engineering. He worked as a post-doctoral in the Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, Arkansas, from March 2015 to March 2016. He is member of the IEEE.

Xiaoqing (Frank) Liu received the PhD degree in computer science from Texas A&M University in College Station, in 1995. He is currently a professor and department head and holds the Rodger S. Kline endowed leadership chair in the Department of Computer Science and Computer Engineering, University of Arkansas in Fayetteville, Arkansas. His current interests include software engineering, service computing, web-based argumentation, and intelligent systems. He published more than 120 refered papers in numerous journals and conferences, such as TWeb, TSC, SPIP, SQJ, ICSE, JSS, ICWS and CSCW. He is member of the IEEE.

Md Mahfuzer Rhaman. He received the BSc degree from Bangladesh University of Engineering and Technology in 2014. He is currently working toward the PhD degree in the Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, Arkansas. His research interests include service discovery and recommendation.

Bing Li received the PhD degree from Computer Science School, Huazhong University of Science and Technology, in 2003. He is currently a professor of Software Engineering Department in the International School of Software, Wuhan University. His current interests include software engineering and service computing. He has published more than 80 papers in well-known conferences and journals. He is member of the IEEE.

Jianxun Liu received the PhD degree in computer science from Shanghai Jiao Tong University, in 2003. He is now a professor in the School of Computer Science and Engineering, Hunan University of Science and Technology. His current interests include service computing and cloud computing. He has published more than 100 papers in well-known conferences and journals.

Mingdong Tang received the PhD degree in computer science from the ICT in the Chinese Academy of Sciences, China, in 2009. He is now a professor in School of Computer Science and Engineering, Hunan University of Science and Technology. His current interests include service computing and social network. He has published more than 60 papers in well-known conferences and journals.