WTCluster: Utilizing Tags for Web Services Clustering

Liang Chen[1], Liukai Hu[1], Zibin Zheng[2], Jian Wu[1], Jianwei Yin[1], Ying Li[1], and Shuiguang Deng[1]

[1] Zhejiang University, China
[2] The Chinese University of Hong Kong, China
{cliang,huliukai,wujian2000,zjuyjw,cnliying,dengsg}@zju.edu.cn, zbzheng@cse.cuhk.edu.hk

Abstract. Clustering web services would greatly boost the ability of web service search engine to retrieve relevant ones. An important restriction of traditional studies on web service clustering is that researchers focused on utilizing web services' WSDL (Web Service Description Language) documents only. The singleness of data source limits the accuracy of clustering. Recently, web service search engines such as Seekda! [1] allow users to manually annotate web services using so called tags, which describe the function of the web service or provide additional contextual and semantical information. In this paper, we propose a novel approach called *WTCluster*, in which both WSDL documents and tags are utilized for web service clustering. Furthermore, we present and evaluate two tag recommendation strategies to improve the performance of *WTCluster*. The comprehensive experiments based on a dataset consists of 15,968 real web services demonstrate the effectiveness of *WTCluster* and tag recommendation strategies.

1 Introduction

A service-oriented computing (SOC) paradigm and its realization through standardized web service technologies provide a promising solution for the seamless integration of single-function applications to create new large-grained and value-added services. SOC attracts industry's attention and is applied in many domains, e.g., workflow management, finances, e-Business, and e-Science. With a growing number of web services, the problem of discovering user required web services is becoming more and more important.

Web service discovery can be achieved by two main approaches: UDDI (*Universal Description Discovery and Integration*) and web service search engines. Recently, the availability of web services in UDDI decreases rapidly as many web service providers decided to publish their web services through their own website instead of using public registries. Al-Masri *et al.* show that more than 53% of the UDDI business registry registered services are invalid, while 92% of web services

[1] http://webservices.seekda.com/

G. Kappel, Z. Maamar, H.R. Motahari-Nezhad (Eds.): ICSOC 2011, LNCS 7084, pp. 204–218, 2011.
© Springer-Verlag Berlin Heidelberg 2011

cached by web service search engines are valid and active [2]. Compared with UDDI, using search engine to search and discover web services becomes more common and effective.

Searching for web services using web service search engines is typically limited to keyword matching on names, locations, businesses, and buildings defined in the web service description file [14]. If the query term does not contain at least one exact word such as the service name, the service is not returned. It is difficult for users to be aware of the concise and correct keywords to retrieve the satisfied services. The keyword-based search mode suffers from low recall, where results containing synonyms or concepts at a higher (or lower) level of abstraction describing the same service are not returned. For example, a service named "*Mobile Messaging Service*" may not be returned from the query term "*SMS*" submitted by the user, even these two keywords are obviously the same at the conceptual level.

To handle the drawbacks of traditional web service search engines, some approaches are proposed. Lim *et al.* propose to make use of ontology to return an expand set of results including subclass, superclass and sibling classes of the concept entered by the user [16]. Elgazzar and Liu *et al.* proposed to handle the drawbacks of traditional search engine by clustering web services based on WSDL documents [6][12]. In their opinion, if web services with similar functionality are placed into the same cluster, more relevant web services could be included in the search result. In this paper, we propose to improve the performance of web service clustering for the purpose of more accurate web service discovery.

Fig. 1. Example of web services' tags

In recent years, tagging, the act of adding keywords (tags) to objects, has become a popular mean to annotate various web resources, e.g., web page bookmarks, online documents, and multimedia objects. Tags provide meaningful descriptions of objects, and allow users to organize and index their contents. Tagging data was proved to be very useful in many domains such as multimedia, information retrieval, data mining, and so on. Recently, a real-world web services search engine Seekda! allows users to manually annotate web services using tags. Figure 1 shows two examples of web services' tags in Seekda!. *MeteorologyWS* [2]

[2] http://www.premis.cz/PremisWS/MeteorologyWS.asmx?WSDL

in Fig. 1(a) is a web service which provides the function of weather forecasting. It has two tags, *weather* and *waether*. However, there is no word *weather* in its service name or WSDL document. Therefore, if a user uses *weather* as his query term, this service will not be retrieved without utilizing the tag information. Besides, the tag *waether* is also useful as some users may make a mistake in the typing process and use *waether* as his query term. Figure 1(b) shows another web service providing car rental information, which is very important for tourists. If we utilize the tag *tourism* in the search engine, this service will be included in the search result about tourism. From these two examples, we can find that the tagging data can help to retrieve more relevant web services.

In this paper, we don't simply use tags to match query terms, but use these tags to improve the performance of web service clustering for the purpose of more accurate web service discovery. In traditional web service clustering, features (e.g., *service name, operation, port*) are extracted from the WSDL document to form a vector, and the similarity between two web services is computed by comparing their corresponding vectors. As the words matching is still needed in the process of similarity computation, the web service *MeteorologyWS* in Fig. 1(a) can hardly be placed into the same cluster with other weather report services which have the word *weather* in their names or WSDL documents. As a service provider, he may have different naming convention and prefers to use *Meteorology* instead of *weather*. However, as a service user, he is likely to annotate the same tag to the services with similar function. Therefore, if we use the tags as part of the vectors to compute the similarities between web services, the performance of web service clustering could be improved. In our proposed *WTCluster* approach, we utilize both WSDL documents and tags, and cluster web services according to a composite similarity generated by integrating tag-level similarity and feature-level similarity between web services. Specifically, we extract 5 features from WSDL document, i.e., *Content*, *Type*, *Message*, *Port*, *Service Name*. To the best of our knowledge, this paper is the first paper to utilize the tagging data to cluster web services.

To evaluate the performance of *WTCluster*, we crawl 15,968 real web services from Seekda!. Through our observation, we find that the performance of *WTCluster* is limited by the web services which have few tags. To handle this problem, we propose two strategies to recommend some relevant tags to the services with few tags. The experiment results in Section 5 show that our tag recommendation strategies improve the performance of *WTCluster*.

In particular, the contribution of this paper can be summarized as follows:

1. We propose a novel web service clustering approach *WTCluster*, in which both WSDL documents and tags are utilized.
2. We propose two tag recommendation strategies to improve the performance of *WTCluster*.
3. We crawl 15,968 real web services to evaluate the performance of *WTCluster* and two tag recommendation strategies.

The rest of this paper is organized as follows: Section 2 highlights the related work of web service discovery and clustering. The detailed calculation of

WTCluster is introduced in Section 3, while two tag recommendation strategies are presented in Section 4. Section 5 shows the experimental results. Finally, Section 6 concludes this paper.

2 Related Work

With the development of service computing and cloud computing, web service discovery is becoming a hot research topic. A lot of work have been done to handle this problem. The approaches for discovering semantic web services and non-semantic web services are different. The semantic-based approaches adopt the formalized description languages such as OWL-S and WSMO for services and develop the reasoning-based similarity algorithms to retrieve the satisfied web services [1][3][10]. High level match-making approaches are usually adopted in the discovery of semantic web services. As non-semantic web services are more popular and supported by the industry circle, we focus on the discovery of non-semantic web services in this paper.

Some non-semantic approaches are proposed to handle the problem of web service discovery in recent years. Xin Dong *et al.* propose to compute the similarity between web services employing the structures of web services (including name, text, operation descriptions, input/output description, etc) [5]. They also propose a search engine called Woogle which supports similarity search for web services. Nayak attempts to handle the service discovery problem by suggesting the current user with other related search terms based on what other users had used in similar queries by using clustering techniques [14]. Nayak proposes to cluster web services based on search sessions instead of individual queries. Songlin Hu *et al.* make use of the content-based publish/subcribe model to handle service discovery problem [7]. Fangfang *et al.* try to reflect the underlying semantics of web services by utilizing the terms within WSDL fully [11]. In Fangfang's work, some external knowledge are firstly employed to compute the semantic distance of terms from two compared services, and then the similarity between two services is measured upon these distances.

Recently, web service clustering is presented as a novel solution to the problem of service discovery. Liu *et al.* propose to extract 4 features, i.e., *content, context, host name, and service name*, from the WSDL document to cluster web services [12]. They take the process of clustering as the preprocessor to discovery, hoping to help in building a search engine to crawl and cluster non-semantic web services. Khalid *et al.* also propose to extract features from WSDL documents to cluster web services [6]. Different from Liu's work, Khalid extracts *content, types, messages, ports, and service name* from WSDL documents.

Although these techniques are relevant, the performances of these approaches are limited by the singleness of source information as they utilize the information in WSDL documents only. In this paper, we propose to utilize both tagging data and WSDL documents to improve the performance of web service clustering. Moreover, we propose two tag recommendation strategies to handle another performance limitation caused by the web services with few tags.

3 WTCluster

In this section, we first describe our proposed framework for web service discovery in Section 3.1, and then introduce feature extraction, similarity computation and integration of *WTCluster* in Section 3.2 and Section 3.3, respectively.

3.1 Framework for Web Service Discovery

Figure 2 shows our proposed framework for web service discovery. This framework consists of two parts: 1) Data Preprocess; 2) Service Discovery. In the first part, WSDL documents and tags of web services are crawled from the Internet and used for clustering. Similar to Khalid's work[6], we extract five important features from WSDL documents, i.e., *Content*, *Type*, *Message*, *Port*, and *Service Name*. After obtaining these five features and tags of web services, we employ our proposed *WTCluster* approach to cluster web services. Since the data preprocess and clustering process is done offline, the efficiency is not a big concern, whereas the accuracy is more important. In the process of service discovery, the user first sends a query term to the web service search engine, and then the search engine returns an expanded search result by retrieving the clustered results.

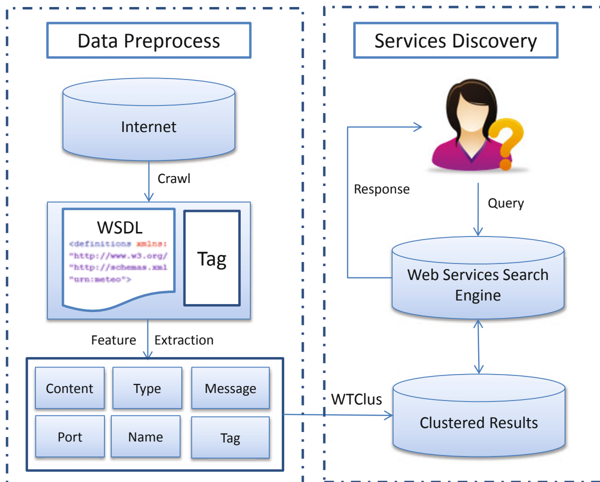

Fig. 2. Framework for web service discovery

3.2 Feature Extraction and Similarity Computation

As discussed above, we extract five features (i.e., *Content, Type, Message, Port, and Service Name*) from web service's WSDL document, and use these five features and tags to cluster web services. In this section, we describe the detailed process of feature extraction, feature-level similarity computation, and tag-level similarity computation.

Content. WSDL document, which describes the function of web service, is actually a XML style document. Therefore, we can use some IR approaches to extract a vector of meaningful content words which can be used as a feature for similarity computation. Our approach for building the content vector consists of four steps:

1. **Building original vector.** In this step, we split the WSDL content according to the white space to produce the original content vector.
2. **Suffix Stripping.** Words with a common stem will usually have the same meaning, for example, *connect*, *connected*, *connecting*, *connection*, and *connections* all have the same stem *connect* [12]. For the purpose of convenient statistics, we strip the suffix of all these words that have the same stem by using a Porter stemmer [15]. Therefore, after the step of suffix stripping, a new content vector is produced, in which words such as *connected* and *connecting* are replaced with the stem *connect.*
3. **Pruning.** In this step, we propose to remove two kinds of words from the content vector. The first kind of word to be removed is XML tag. For example, the words *s:element*, *s:complexType*, and *wsdl:operation* are XML tags which are not meaningful for the comparison of content vector. As the XML tags used in a WSDL document are predefined, it is easy to remove them from the content vector. Content words are typically nouns, verbs or adjectives, and are often contrasted with function words which have little or no contribution to the meanings of texts. Therefore, the second kind of word to be removed is function word. Church *et al.* stated that the function words can be distinguished from contents words using a Poisson distribution to model word occurrence in documents [9]. Typically, a way to decide whether a word w in the content vector is a function word is computing the degree of overestimation of the observed document frequency of the word w, denoted by n_w using Poisson distribution. The overestimation factor can be calculated as follows.
$$\Lambda_w = \frac{\hat{n_w}}{n_w}, \tag{1}$$
where $\hat{n_w}$ is the estimated document frequency of the word w. Specifically, the word with higher value of Λ_w has higher possibility to be a content word. In this paper, we set a threshold Λ_T for Λ_w ,and take the words which have Λ_w higher than threshold as content words. The value of threshold Λ_T is as follows.
$$\Lambda_T = \begin{cases} avg[\Lambda] & \text{if}(avg[\Lambda] > 1); \\ 1 & \text{otherwise} \end{cases} \tag{2}$$
where $avg[\Lambda]$ is the average value of the observed document frequency of all words considered. After the process of pruning, we can obtain a new content vector, in which both XML tags and function words are removed.
4. **Refining.** Words with very high occurrence frequency are likely to be too general to discriminate between web services. After the step of pruning, we implement a step of refining, in which words with too general meanings are

removed. Clustering based approaches were adopted to handle this problem in some related work [12][6]. In this paper, we choose a simple approach by computing the frequencies of words in all WSDL documents and setting a threshold to decide whether a word has to be removed.

After the above 4 steps, we can obtain the final content vector. In this paper, we use NGD (Normalized Google Distance) [4] to compute the content-level similarity between two web services. Given web services s_1, s_2, and their content vector $content_{s_1}$, $content_{s_2}$, the detailed equation for content-level similarity computation is as follows.

$$Sim_{content}(s_1, s_2) = \frac{\sum_{w_i \in content_{s_1}} \sum_{w_j \in content_{s_2}} sim(w_i, w_j)}{|content_{s_1}||contents_2|}, \quad (3)$$

where $|content_{s_1}|$ means the cardinality of $content_{s_1}$, the equation for computing the similarity between two words is as follows.

$$sim(w_i, w_j) = 1 - NGD(w_1, w_2) \quad (4)$$

In (4), we compute the similarity between two words using NGD based on the word co-existence in web pages. Due to space limitation, we don't introduce the detailed computation of NGD. As the number of words left in the content vector is limited after above 4 steps, the time cost for content-level similarity computation can be accepted.

Type. In a WSDL document, each input and output parameter contains a name attribute and a type attribute. Sometimes, parameters may be organized in a hierarchy by using complex types. Due to different naming conventions, the name of parameter is not always a useful feature, whereas the type attribute which can partially reflect the service function is a good candidate feature.

As Fig. 3 shows, the type of element *ProcessForm* (we name it $type_1$) is a complextype which has 5 parameters: *FormData* (string), *FormID* (int), *GroupID* (int), *szPageName* (string), and *nAWSAccountPageID* (int). If another service s_2 has a complextype $type_2$ which also contains 2 string type parameters and 3 int type parameters, we say $type_1$ and $type_2$ are matched. Specifically, in the process of type matching, the order of parameters in the complextype is not considered. We therefore extract the defined types, count the number of different types in the complextype, and compute the type-level similarity between two services using following equation.

$$Sim_{type}(s_1, s_2) = \frac{2 \times Match(Type_{s_1}, Type_{s_2})}{|Type_{s_1}| + |Type_{s_2}|}, \quad (5)$$

where $Type_{s_1}$ means the set of defined types in $s_1's$ WSDL document, $Match$ $(Type_{s_1}\ , Type_{s_2})$ means the number of matched types between these two services, and $|Type_{s_1}|$ means the cardinality of $Type_{s_1}$.

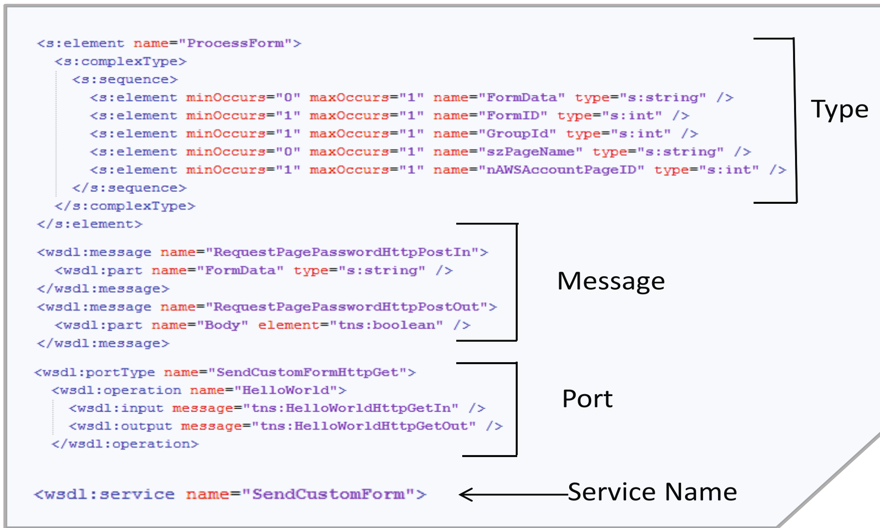

Fig. 3. Types, Message, Port, Service Name in WSDL document

Message. Message is used to deliver parameters between different operations. One message contains one or more parameters, and one parameter is associated with one type as we discussed above. Message definition is typically considered as an abstract definition of the message content, as the name and type of the parameter contained in the message are presented in the message definition. Fig. 3 shows two simple message definitions. In the first definition, the message named as *RequestPagePasswordHttpPostIn* contains one parameter *FormData* which is a *string* type. In the second definition, the message *RequestPagePasswordPostOut* contains one parameter *Body* whose type is a complextype named as *tns:boolean*. Similar to (5), we match the messages' structures to compute the message-level similarity between web services.

Port. The portType element combines multiple message elements to form a complete one-way or round-trip operation. Figure 3 shows an example of portType *SendCustomFormHttpGet* which contains some operations (due to space limitation, we only list one operation in this portType). As the portType consists of some messages, we can get the match result of portType according to the match result of messages. Similar to the computation of type-level and message-level similarity, we also use (5) to compute the port-level similarity.

Service Name. As the service name (*sname*) can partially reflect the service function, it is an important feature in WSDL document. Before computing the *sname*-level similarity, we first implement a word segmentation process to service name. For example, the service name *SendCustomForm* in Fig. 3 can be separated into three words *Send*, *Custom*, and *Form*. A simple version of word segmentation is splitting the service name according to the capital letters.

However, the performance of this simple version is not satisfied due to different naming conventions. In this paper, we first use this simple version to split the service name, and then manually adjust the final result. After the process of word segmentation, $s_1's$ name $SName_{s_1}$ can be presented as a set of words. And then we can use (3) and (4) to compute the *sname*-level similarity between web services.

Tag. The tagging data of web services describes the function of web services or provide additional contextual and semantical information. In this paper, we propose to improve the performance of traditional WSDL-based web service clustering by utilizing the tagging data. Given a web service s_i contains three tags t_1, t_2, t_3, we name the tag set of s_i as $T_i = \{t_1, t_2, t_3\}$. According to the Jaccard coefficient [8] method, we can calculate the tag-level similarity between two web services s_i and s_j as follows:

$$Sim_{tag}(s_i, s_j) = \frac{|T_i \cap T_j|}{|T_i \cup T_j|}, \tag{6}$$

where $|T_i \cap T_j|$ means the number of tags that are both annotated to s_i and s_j, and $|T_i \cup T_j|$ means the number of unique tags in set T_i and T_j, i.e., $|T_i \cup T_j| = |T_i| + |T_j| - |T_i \cap T_j|$.

3.3 Similarity Integration

In *WTCluster*, we use K-Means [13] clustering approach to cluster web services. K-Means is a widely adopted clustering algorithm which is simple and fast. The drawback of this algorithm is that the number of clusters has to be predefined manually before clustering. According to the six similarities calculated above, the composite similarity $CSim(s_i, s_j)$ between web services s_i and s_j is as follows:

$$CSim(s_i, s_j) = (1 - \lambda)Sim_{wsdl}(s_i, s_j) + \lambda Sim_{tag}(s_i, s_j), \tag{7}$$

where λ is the weight of the tag-level similarity, and the $Sim_{wsdl}(s_i, s_j)$ is the WSDL-level similarity which consists of five feature-level similarities between two services. The range of the value of λ is [0,1]. When $\lambda \in (0, 1)$, $CSim(s_i, s_j)$ is equal to 1 if the WSDL documents and tags of these two services are identical, and $CSim(s_i, s_j)$ is equal to 0 if both the WSDL documents and the tags of these two services are completely different. Specifically, *WTCluster* is equal to WSDL-based web service clustering approach when $\lambda = 0$, while *WTCluster* clusters web services only according to the tag-level similarity when $\lambda = 1$. We measure the WSDL-level similarity between web services s_i and s_j as follows:

$$\begin{aligned} Sim_{wsdl}(s_i, s_j) = w_1 Sim_{content}(s_i, s_j) + w_2 Sim_{type}(s_i, s_j) + w_3 Sim_{message}(s_i, s_j) \\ + w_4 Sim_{port}(s_i, s_j) + w_5 Sim_{sname}(s_i, s_j), \end{aligned} \tag{8}$$

where w_1, w_2, w_3, w_4, and w_5 are the user-defined weights of *Content, Type, Message, Port*, and *Service Name*, respectively. In particular, $w_1 + w_2 + w_3 + w_4 + w_5 = 1$.

4 Tag Recommendation

After examining the tagging data crawled from the Internet, we find the distribution of tags is not uniform. Some web services have more than 10 tags, while some ones have only 1 or 2 tags. As we compute the tag-level similarity by matching the common tags between two services, the web services with few tags lowers down the value of tag-level similarity. In this section, we propose to handle this problem by recommending a set of relevant tags to the web services with few tags.

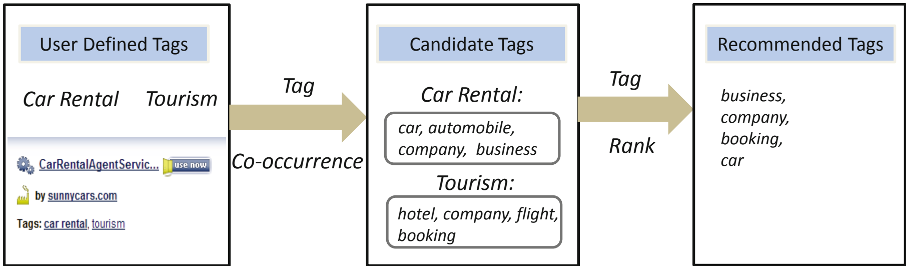

Fig. 4. Overview of Tag Recommendation Process

Figure 4 show the overview of tag recommendation process. From this figure, we can find that the process of tag recommendation can be divided into two steps. Specifically, we collect all annotated tags before the process of tag recommendation. In the first step, we first compute the co-occurrence between the user defined tags and any other tags, and then select the top-k co-occurrent tags of each user defined tag as the candidate tags. In Fig. 4, the number of k is set as 4, and the top-4 co-occurrent tags of *Tourism* are *hotel, company, flight,* and *booking*. There are some approaches to compute the co-occurrence, and we propose to use Jaccard coefficient method[8] in this paper. The detailed equation is as follows.

$$Co(t_i, t_j) = \frac{|t_i \bigcap t_j|}{|t_i \bigcup t_j|}, \tag{9}$$

where $|t_i \bigcap t_j|$ means the number of web services that have both t_i and t_j, and $|t_i \bigcup t_j|$ means the number of web services that have t_i or t_j. After the first step, for each user defined tag $u \in U$ (U is the set of user defined tags), we can get a list of candidate tags C_u.

In the second step, we rank the candidate tags and select the top-k tags as the recommended tags. In this paper, we propose two strategies to rank candidate tags.

Vote. In the *Vote* strategy, we use the idea of voting to compute a score for each candidate tag $c \in C$ (C is the set of all candidate tags). Given a candidate

tag c, we first use (10) to compute the value of $vote(u, c)$ between tag c and each user defined tag $u \in U$.

$$vote(u,c) = \begin{cases} 1 & \text{if c} \in C_u \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

After obtaining the voting result from each user defined tag, we count the voting results to get the final score by using (11).

$$score(c) = \sum_{u \in U} vote(u, c) \tag{11}$$

After obtaining all final scores, we rank the candidate tags to get the top-k recommended tags.

Sum. In the *Sum* strategy, we compute the score of the candidate tag c by summing the value of co-occurrence between c and each user defined tag u. The detailed equation is as follows.

$$score(c) = \sum_{u \in U} Co(u, c), \tag{12}$$

where the value of $Co(u, c)$ can be computed by using (9).

5 Experiment

In this section, we first compare the performances of different web service clustering approaches and then study the performances of two tag recommendation strategies .

5.1 Experiment Setup

To evaluate the performance of web service clustering approaches and tag recommendation strategies, we crawl 15,968 real web services form the web service search engine Seekda!. For each web service, we get the data of service name, WSDL document, tags, availability, and the name of service provider.

All experiments are implemented with JDK 1.6.0-21, Eclipse 3.6.0. They are conducted on a Dell Inspire R13 machine with an *2.27 GHZ Intel Core I5 CPU* and *2GB RAM*, running *Windows7 OS*.

5.2 Performance of Web Service Clustering

As the manual creation of ground truth costs a lot of work, we randomly select 200 web services from the dataset we crawled to evaluate the performance of web service clustering. We perform a manual classification of these 200 web services

to serve as the ground truth for the clustering approaches. Specifically, we distinguish the following categories: "HR", "On Sale", "Tourism", and "University". There are 31 web services in "HR" category, 26 web services in "On Sale" category, 32 web services in "Tourist" category, and 27 web services in "University" category. Due to the space limitation, we don't shows the detailed information of these web services. To evaluate the performance of web service clustering, we introduce two metrics (Precision and Recall) which are widely adopted in the Information Retrieval domain.

$$Precision_{c_i} = \frac{succ(c_i)}{succ(c_i) + mispl(c_i)}, Recall_{c_i} = \frac{succ(c_i)}{succ(c_i) + missed(c_i)}, \quad (13)$$

where $succ(c_i)$ is the number of services successfully placed into cluster c_i, $mispl(c_i)$ is the number of services that are incorrectly placed into cluster c_i, and $missed(c_i)$ is the number of services that should be placed into c_i but are placed into another cluster.

In this section, we compare the performances of three web service clustering approaches:

1. ***WCluster.*** In this approach, web services are clustered only according to the WSDL-level similarity between web services (calculated in (8)). This approach was adopted in some related work [6][12].
2. ***WTCluster^1.*** In this approach, we utilize both the WSDL documents and the tagging data, and cluster the web services according to the composite similarity calculated in (7).
3. ***WTCluster^2.*** In this approach, we first implement the tag recommendation process and then cluster web services using $WTCluster^1$ approach. In addition, we use the *Vote* strategy in this experiment.

Figure 5 shows the performance comparison of above 3 web service clustering approaches. For simplicity, we set $w_1 = w_2 = w_3 = w_4 = w_5 = 0.2$ and $\lambda = 0.5$. From Fig. 5, we can observe that our proposed *WTCluster* approaches ($WTCluster^1$, $WTCluster^2$) outperform the traditional *WCluster* approach both in the comparison of precision and recall. As we discussed above,

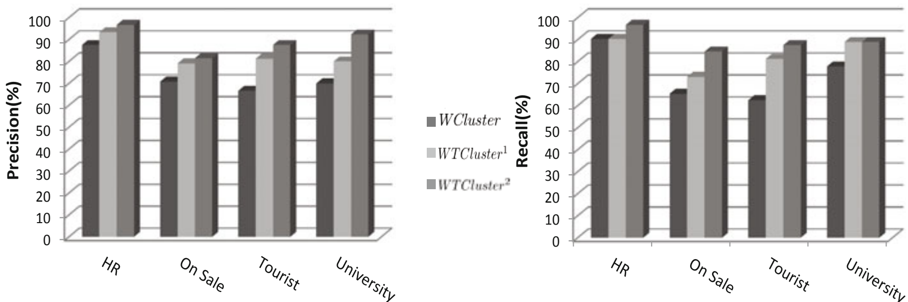

Fig. 5. Performance comparison of three web service clustering approaches

the tags of web services contains a lot of information, such as service function, location, and other semantical information. Utilizing these information improves the performance of web service clustering. Moreover, it can be observed that the approach $WTCluster^2$ which contains the process of tag recommendation outperforms the $WTCluster^1$ approach. It demonstrates that adding relevant tags to web services which have few tags can improve the performance of *WTCluster* approach.

5.3 Evaluation of Tag Recommendation Strategies

Before evaluating the performance of tag recommendation, we select 1,800 web services which contain 1254 unique tags as the dataset for evaluation. The ground truth is manually created through a blind review pooling method, where for each of the 1800 web services, the top 10 recommendations from each of the two strategies were taken to construct the pool. The volunteers were then asked to evaluate the descriptiveness of each of the recommended tags in context of the web services. We provide the WSDL documents and web service descriptions to volunteers to help them. The volunteers were asked to judge the descriptiveness on a three-point scale: *very good*, *good*, *not good*. The distinction between *very good* and *good* is defined to make the assesment task conceptually easier for the user. Finally, we get 212 *very good* judgements (16.9%), 298 *good* judgements (23.7%), and 744 *not good* judgements (59.4%).

To evaluate the performance of tag recommendation, we adopt two metrics which capture the performance at different aspects:

– **Success at rank K (S@K).** The success at rank K is defined as the percentage of *good* or *very good* tags take in the top K recommended tags, averaged over all judged web services.
– **Precision at rank K (P@K).** Precision at rank K is defined as the proportion of retrieved tags that is relevant, averaged over all judged web services.

Table 1 shows the S@K comparison of our proposed two recommendation strategies, where the *Given Tag* means the number of tags that the target web service has. Take the Sum strategy as example, when *Given Tag* varies from 1 to 2, the average value of S@K is over 0.7, which means that more than 70% recommended tags have *good* or *very good* descriptiveness. From Table 1, it can be observed that when *Given Tag* vary from 1 to 2, the performance of *Sum* strategy is better than the performance of *Vote* strategy in terms of S@K, while the performance of *Vote* strategy is better when *Given Tag* is larger than 5.

Table 2 shows the comparison of two tag recommendation strategies in terms of P@K. From Table 2, it can be observed that the value of P@K decreases when *Given Tag* increases. This is because the number of relevant tags to one certain web service is limited. When *Given Tag* increases, the number of left relevant tags decreases, which leads to the decrease of P@K. In addition, P@K achieves its largest value when K=1, and decreases when the value of K increases. It can be found that the *Vote* strategy basically outperforms the *Sum* strategy in terms of P@K.

Table 1. S@K comparison of two tag recommendation strategies

Given Tag	Method	K=1	K=2	K=3	K=4	K=5
1-2	Sum	**0.8132**	**0.7081**	**0.6738**	**0.7087**	**0.7181**
	Vote	0.6392	0.5949	0.6737	0.7005	0.6972
3-5	Sum	0.7534	0.7143	0.7380	0.6852	0.6720
	Vote	0.7867	0.6646	0.7042	0.7022	0.7103
>5	Sum	0.7632	0.7211	0.6944	0.6975	0.6647
	Vote	**0.8136**	**0.7769**	**0.7749**	**0.7262**	**0.6973**

Table 2. P@K comparison of two tag recommendation strategies

Given Tag	Method	K=1	K=2	K=3	K=4	K=5
1-2	Sum	0.6933	0.5083	0.4277	0.3788	0.3562
	Vote	**0.7879**	**0.5495**	**0.4503**	**0.3947**	**0.3689**
3-5	Sum	0.6512	0.4857	0.4171	0.3654	0.3345
	Vote	**0.7415**	**0.5414**	**0.4496**	**0.3925**	**0.3494**
>5	Sum	0.5894	0.4656	**0.4365**	0.3451	0.3508
	Vote	**0.7148**	**0.5478**	0.4105	**0.4026**	**0.3658**

6 Conclusion

In this paper, we propose to utilize the tagging data to improve the performance of web service clustering for the purpose of more accurate web service discovery. In our proposed *WTCluster* approach, we first extract five features from the WSDL document and compute the WSDL-level similarity between web services. Then, we use K-means algorithm to cluster web services according to the composite similarity which is integrated by WSDL-level similarity and tag-level similarity. To evaluate the performance of web service clustering, we crawl 15,968 real web services from the web service search engine Seekda!. The experimental results show that *WTCluster* outperforms the traditional WSDL-based approach.

Moreover, we propose two tag recommendation strategies to attack the performance limitation of *WTCluster* caused by the web services with few tags. The experiments based on real web services demonstrates that the tag recommendation process improves the performance of *WTCluster*.

Acknowledgements. This research is was partially supported by the National Technology Support Program under grant of 2011BAH15B05, the National Natural Science Foundation of China under grant of 61173176, Science and Technology Program of Zhejiang Province under grant of 2008C03007, National High-Tech Research and Development Plan of China under Grant No. 2009AA110302, National Key Science and Technology Research Program of China (2009ZX01043-003-003).

References

1. Agarwal, S., Studer, R.: Automatic matchmaking of web services. In: International Conference on Web Services, pp. 45–54 (2006)
2. Al-Masri, E., Mahmoud, Q.H.: Investigating web services on the world wide web. In: International World Wide Web Conference, pp. 795–804 (2008)
3. Benatallah, B., Hacid, M., Leger, A., Rey, C., Toumani, F.: On automating web services discovery. The VLDB Journal 14(1), 84–96 (2005)
4. Cilibrasi, R.L., Vitnyi, P.M.B.: The google similarity distance. IEEE Transactions on Knowledge and Data Engineering 19(3), 370–383 (2007)
5. Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang, J.: Similarity search for web services. In: International Conference on Very Large Data Bases, pp. 372–383 (2004)
6. Elgazzar, K., Hassan, A.E., Martin, P.: Clustering wsdl documents to bootstrap the discovery of web services. In: International Conference on Web Services, pp. 147–154 (2009)
7. Hu, S., Muthusamy, V., Li, G., Jacobsen, H.A.: Distributed automatic service composition in large-scale systems. In: Proc. of Distributed Event-Based Systems Conference, pp. 233–244 (2008)
8. Jain, A., Dubes, R.: Algorithms for Clustering Data. Prentice Hall, New Jersey (1988)
9. Church, K., Gale, W.: Inverse document frequency (idf): a measure of deviations from poisson. In: Proceedings of the ACL 3rd workshop on Very Large Corpora, pp. 121–130 (1995)
10. Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with owls-mx. In: International Conference on Autonomous Agents and Multiagent Systems, pp. 915–922 (2006)
11. Liu, F., Shi, Y., Yu, J., Wang, T., Wu, J.: Measuring similarity of web services based on wsdl. In: International Conference on Web Services, pp. 155–162 (2010)
12. Liu, W., Wong, W.: Web service clustering using text mining techniques. International Journal of Agent-Oriented Software Engineering 3(1), 6–26 (2009)
13. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Proc. of the Fifth Symposium on Math, Statistics, and Probability, pp. 281–297 (1967)
14. Nayak, R.: Data mining in web service discovery and monitoring. International Journal of Web Services Research 5(1), 62–80 (2008)
15. Porter, M.F.: An algorithm for suffix stripping. Program. 14(3), 130–137 (1980)
16. Lim, S.-Y., Song, M.-H., Lee, S.-J.: The Construction of Domain Ontology and its Application to Document Retrieval. In: Yakhno, T. (ed.) ADVIS 2004. LNCS, vol. 3261, pp. 117–127. Springer, Heidelberg (2004)
17. Zhang, Y., Zheng, Z., Lyu, M.R.: Wsexpress: A qos-aware search engine for web services. In: International Conference on Web Services, pp. 91–98 (2010)