Knowl Inf Syst (2014) 38:207–229
DOI 10.1007/s10115-013-0623-0

REGULAR PAPER

# Clustering Web services to facilitate service discovery

**Jian Wu · Liang Chen · Zibin Zheng ·
Michael R. Lyu · Zhaohui Wu**

Received: 16 December 2011 / Revised: 20 October 2012 / Accepted: 20 October 2012 /
Published online: 17 March 2013

**Abstract** Clustering Web services would greatly boost the ability of Web service search engine to retrieve relevant services. The performance of traditional Web service description language (WSDL)-based Web service clustering is not satisfied, due to the singleness of data source. Recently, Web service search engines such as *Seekda!* allow users to manually annotate Web services using tags, which describe functions of Web services or provide additional contextual and semantical information. In this paper, we cluster Web services by utilizing both WSDL documents and tags. To handle the clustering performance limitation caused by uneven tag distribution and noisy tags, we propose a hybrid Web service tag recommendation strategy, named *WSTRec*, which employs tag co-occurrence, tag mining, and semantic relevance measurement for tag recommendation. Extensive experiments are conducted based on our real-world dataset, which consists of 15,968 Web services. The experimental results demonstrate the effectiveness of our proposed service clustering and tag recommendation strategies. Specifically, compared with traditional WSDL-based Web service clustering approaches, the proposed approach produces gains in both precision and recall for up to 14 % in most cases.

**Keywords** Web service · Clustering · Tag recommendation · Service discovery

J. Wu (✉) · L. Chen · Z. Wu
College of Computer Science, Zhejiang University, Hangzhou, China
e-mail: wujian2000@zju.edu.cn

L. Chen
e-mail: cliang@zju.edu.cn

Z. Wu
e-mail: wzh@zju.edu.cn

Z. Zheng · M. R. Lyu
Department of Computer Science and Engineering, Shenzhen Research Institute,
The Chinese University of Hong Kong, New Territories, Hong Kong
e-mail: zbzheng@cse.cuhk.edu.hk

M. R. Lyu
e-mail: lyu@cse.cuhk.edu.hk

## 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, discovering the user-required Web services is becoming more and more important.

Web service discovery can be achieved by two main approaches: universal description discovery and integration (UDDI) 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 Web site instead of using public registries. Al-Masri et al. [1] show that more than 53 % of registered services in UDDI business registries are invalid, while 92 % of Web services cached by Web service search engines are valid and active. Compared with UDDI, using search engine to discover Web services becomes common and effective.

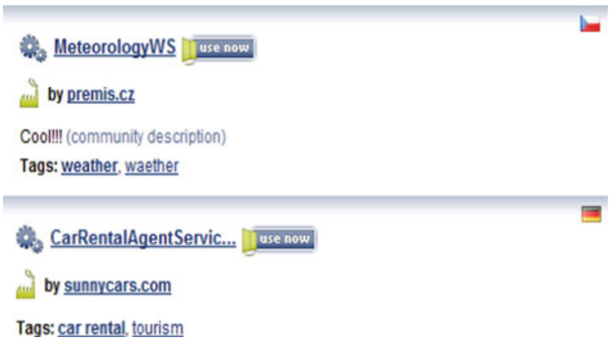
In the field of service computing, Web service search is typically limited to keyword matching on names, locations, businesses, and buildings defined in the Web service description file [2]. If the query term does not contain at least one exact word such as the service name, the service is not returned. It is difficult for users to be aware of the concise and correct keywords to retrieve the targeted services satisfactorily. 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 the keyword-based Web service search engines, some approaches are proposed to extend the search result. Lim et al. [3] propose to make use of ontology to return an expanded set of results including subclass, superclass, and sibling classes of the concept entered by the user, while Elgazzar et al. [4] and Liu et al. [5] propose to handle the drawbacks of traditional search engine by clustering Web services based on WSDL documents. In these previous works, if Web services with similar functionality are placed into the same cluster, then more relevant Web services could be included in the search result. In fact, many experiments have demonstrated that good Web service clustering boosts the performance of Web service search. In this paper, we attempt to improve the performance of Web service clustering for the purpose of more accurate Web service discovery.

Although WSDL-based Web service clustering approaches improve the precision and recall, the search result is still not satisfactory. Recently, tagging, the act of adding keywords (tags) to objects, has become a popular means to annotate various Web 2.0 objects, 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 [6]. In fact, tags have been demonstrated as one of the best textual features to be exploited by various information retrieval (IR) services, such as automatic object classification and clustering. Recently, a real-world Web services search engine Seekda![1] allows users to manually associate tags with Web services. Figure 1 shows two examples of tags of Web services in Seekda!. *MeteorologyWS*[2] in Fig. 1a is a Web service providing weather forecasting service and is associated

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

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


MeteorologyWS use now

by premis.cz

Cool!!! (community description)

Tags: weather, waether

CarRentalAgentServic... use now

by sunnycars.com

Tags: car rental, tourism

**Fig. 1** Two Web services annotated with tags in Seekda!

with two tags, i.e., *weather* and *waether* (wrong spelling). However, there is no word *weather* in its service name or WSDL document, due to the naming convention of the service provider. Therefore, if users use *weather* as their query term, this service will not be retrieved if only WSDL document is utilized. *CarRentalAgentService*[3] in Fig. 1b is a Web service providing car rental information and is associated with tags *car rental* and *tourism*. If we utilize the tagging data, this service may not only be included in the search result about car rental, but also be recommended to the users who need Web services about tourism. From these two cases, we can find that the tagging data are helpful in clustering functionality-similar services and achieving more relevant services.

In contrast to previous work [4,5] in which only WSDL documents are employed, we jointly exploit both WSDL documents and tags for Web service clustering. Specifically, we take advantage of previous WSDL-based clustering approach [4], extracting five features from WSDL documents, i.e., *Content, Type, Message, Port, and Service Name* and computing the WSDL-level similarities among Web services. Further, we compute the tag-level similarities among Web services. Finally, WSDL-level similarity and tag-level similarity are merged to be a composite similarity, which is used to cluster Web services.

When employing tags to help Web service clustering, there are two main problems, which limit the effectiveness of tag data in Web service clustering: (1) Uneven distribution: some Web services have more than 20 tags, while some have only 1 or even no tag; and (2) Noise: tags are created by end users and thus may contain noises (e.g., misspellings or unrelated terms). To handle the first problem, we propose a hybrid tag recommendation strategy, named *WSTRec*, which employs tag co-occurrence [7], tag mining, and relevance measurement for tag recommendation. Specifically, tag co-occurrence is employed to suggest candidate tags for Web services by extracting association rules among existing tags, while the tag mining technique is used to mine candidate tags from WSDL textual features (e.g., service description and service name) for Web services with no tags. Relevance metrics are adopted to rank the candidate tags in the process of tag recommendation and filter noisy tags or reduce their importance. By employing *WSTRec*, tagging data are smoothed, which is helpful for more accurate Web service clustering.

To evaluate the performance of the proposed web service clustering approach and tag recommendation strategy, we crawl 15,968 real Web services from Seekda! Specifically, for each service, we crawl service name, service description, WSDL document, and tags.

[3] http://rpc.test.sunnycars.com/CarRentalAgentService121/CarRentalAgentService.asmx?wsdl.

Experimental results show that our proposed clustering approach produces gains in both precision and recall for up to 14 % in most cases.

The contribution of this paper can be summarized as follows:

- We propose a Web service clustering approach, which improves the performance of Web service clustering by utilizing both WSDL documents and Web service tags.
- We propose a hybrid tag recommendation strategy to attack the service clustering performance limitation caused by uneven tag distribution and noisy tags.
- We crawl 15,968 real Web services to evaluate the performance of Web service clustering and tag recommendation strategy.

The rest of this paper is organized as follows: Sect. 2 introduces the related work of Web service discovery and clustering. Section 3 presents the architecture of Web service clustering. The detailed processes of data preprocessing and Web service clustering are introduced in Sect. 4, while tag recommendation strategy is presented in Sect. 5. Section 6 shows the experimental results and Sect. 7 concludes this paper.

## 2 Related work

With the development of service computing, searching Web services according to users' requirements, i.e., Web service discovery [8–11], is becoming a hot research topic. Much work has been done to address it. Web services can be classified into two main kinds, i.e., semantic and non-semantic, based on the description language. Approaches for discovering semantic and non-semantic Web services are different. The semantic-based approaches adopt the formalized description languages such as OWL-S[4] and WSMO[5] for services and develop the reasoning-based similarity algorithms to retrieve the satisfied Web services [12,13]. High level match-making approaches are usually adopted in the discovery of semantic Web services. Specifically, Boualem Benatallah et al. [14] propose to tackle the problem of service discovery in the context of description logics.

There are a number of approaches proposed in recent years for non-semantic Web service discovery [2,15–23]. Dong et al. [16] propose to compute the similarity between Web services by employing the structures of Web services (including name, text, operation descriptions, and input/output description). They also propose a search engine called Woogle[6] which supports similarity search for Web services. Nayak [2] attempts to handle the problem by suggesting to the current user with other related search terms based on what other users had used in similar queries by applying clustering techniques. Web services are clustered based on search sessions instead of individual queries. Hu et al. [17] make use of the content-based publish/subscribe model to handle the problem. Fangfang et al. [18] try to reflect the underlying semantics of Web services by utilizing the terms within WSDL fully. In their work, external knowledge is firstly employed to compute the semantic distance of terms from two compared services and then the similarity between them is measured upon these distances.

Recently, Web service clustering [24,25] is proved as an effective solution to boost the performance of Web service discovery. The most widely employed approach for it is similarity based, including (1) semantic based and (2) non-semantic based. Ontology is utilized to

[4] http://www.w3.org/Submission/OWL-S/.

[5] http://www.w3.org/Submission/WSMO/.

[6] http://db.cs.washington.edu/woogle.html.

compute the semantic similarity between Web services in many studies [26–29]. Specifically, Cristina et al. [28] propose to use an ant-based method to cluster Web services based on semantic similarity. In this paper, we focus on the clustering of non-semantic Web services, as such services are more popular and more widely supported by the industry circle.

For the calculation of non-semantic similarity between Web services, WSDL-based approaches are the most representative work [4,5,30]. Liu et al. [5,30] propose to extract 4 features, i.e., *content, context, host name, and service name*, from the WSDL document to cluster Web services. 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. [4] also propose to extract features from WSDL documents to cluster Web services. Different from the work [5,30], Khalid extracts *content, types, messages, ports, and service name* from WSDL documents.

Although WSDL-based techniques are widely adopted, the performance is rather limited since only WSDL documents are employed. With the development of Web service community, more and more tags are annotated to Web services by users. These tags can be employed to enhance the accuracy of service discovery. However, limited work has exploited tagging data for service discovery. In our preliminary work [31], we investigated the benefits of utilizing both WSDL documents and tagging data to cluster Web services. The findings motivate our present study. In this paper, we improve the performance of Web service clustering by introducing a new WSDL-level similarity computation mechanism and by proposing a hybrid Web service tag recommendation strategy to attack the problems of uneven tag distribution and noisy tags.

## 3 Architecture of Web service clustering

Figure 2 shows the architecture of our proposed Web service clustering approach. It consists of three modules: (1) data preprocessing, (2) WSTRec (i.e., Web service tag recommendation), and (3) Web services clustering. In the first module, WSDL documents and tags of Web services are crawled from the Internet. Similar to the work in [4], we extract five important features from WSDL documents, i.e., *Content*, *Type*, *Message*, *Port*, and *Service Name*. Thus, we can obtain tagging data and five kinds of feature data for Web service clustering after data preprocessing.

These data are sent to the module of Web service clustering, in which feature-level similarities and tag-level similarities are integrated into a global similarity to cluster Web services. Since the data preprocessing, WSTRec, and Web service clustering modules are executed off-line, the efficiency is not a big concern, whereas the accuracy is more important.

When employing tags to help Web service clustering, there are two main problems which limit the effectiveness of tag data in Web service clustering: (1) uneven distribution and (2) noise. To address these problems, we introduce WSTRec strategy, in which tag co-occurrence, tag mining, and relevant metrics are adopted. Specially, WSTRec is implemented to do some process on the tagging data. Thus, the module of WSTRec can also be treated as data preprocessing.

## 4 Web service clustering

In this section, we first describe the process of feature extraction and the corresponding feature-level similarity computation, including tagging data and five features extracted from

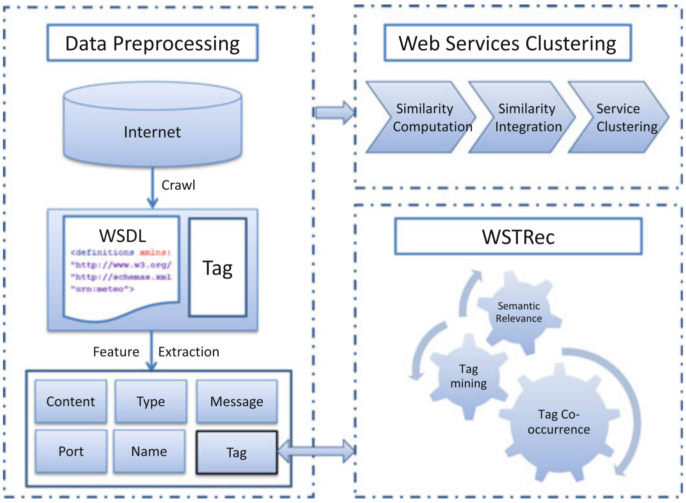


**Fig. 2** Architecture of Web service clustering

a WSDL document. Then, we introduce similarity integration and service clustering. For the convenience of our readers, Table 1 summarizes important notations used in this paper.

## 4.1 Feature extraction and similarity computation

Five features (i.e., *Content, Type, Message, Port, and Service Name*) are extracted from a Web service's WSDL document. These five features and tags are employed to cluster Web services. In the following, we describe the detailed process of feature extraction and the corresponding similarity computation.

### *4.1.1 Content*

A WSDL document, which describes the function of Web service, is an XML style document. Therefore, we can use 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 an 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* [5]. For the purpose of convenient statistics, we strip the suffix of all these words that have the same stem by using a Porter stemmer [32]. 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 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 vectors. 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

**Table 1** Key notations and their descriptions

| Notation | Definition and brief description |
|---|---|
| $Sim_{feature}(s_1, s_2)$ | Feature-level similarity between Web services $s_1$ and $s_2$, in Eqs. (3) and (6) |
| $Sim_{wsdl}(s_1, s_2)$ | WSDL-level similarity between Web services $s_1$ and $s_2$, in Eq. (8) |
| $Sim_{tag}(s_1, s_2)$ | Tag-level similarity between Web services $s_1$ and $s_2$, in Eq. (7) |
| $GSim(s_1, s_2)$ | Global similarity between Web services, integrated by tag-level similarity and WSDL-level similarity, in Eq. (9) |
| $Sim(w_i, w_j)$ | Semantic similarity between words $w_i$ and $w_j$, in Eq. (4) |
| $NGD(w_i, w_j)$ | Normalized Google Distance between words $w_i$ and $w_j$, in Eq. (5) |
| $w_1, \ldots, w_5$ | Weights of feature-level similarities in WSDL-level similarity, in Eq. (8) |
| $\lambda$ | Weight to balance tag-level similarity and WSDL-level similarity in the global similarity, in Eq. (9) |
| $TF-IDF_{t,s_1}$ | The TF-IDF value of term $t$ in the Web service $s_1$, in Eq. (10) |
| $\mathcal{T}$ | Training data in the tag recommendation |
| $\mathcal{P}$ | Testing data in the tag recommendation |
| $\theta$ | Strength of the extracted association rule |
| $NR(n)$ | The number of association rules extracted from $n$ tags |
| $\Phi$ | Score of candidate tag based on tag co-occurrence |
| $\Omega$ | Score of candidate tag based on semantic relevance |
| $\delta$ | Weight to balance $\Phi$ and $\Omega$ in the final tag ranking |
| $\mathcal{N}$ | Number of web services in one cluster |
| $\mathcal{C}$ | Number of clusters |
| $\mathcal{G}$ | Number of initial tags that the target Web service has |
| $K$ | Number of recommended tags |

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. [33] state that the function words could be distinguished from content words using a Poisson distribution to model word occurrence in documents. Typically, a way to decide whether a word $w$ in the content vector is a function word is by 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 $\Lambda_w$ has higher possibility to be a content word. In this paper, we set a threshold $\Lambda_T$ for $\Lambda_w$ and take the words which have $\Lambda_w$ higher than threshold as content words. The value of threshold $\Lambda_T$ is as follows.

$$\Lambda_T = \begin{cases} avg[\Lambda] & \text{if}(avg[\Lambda] > 1); \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

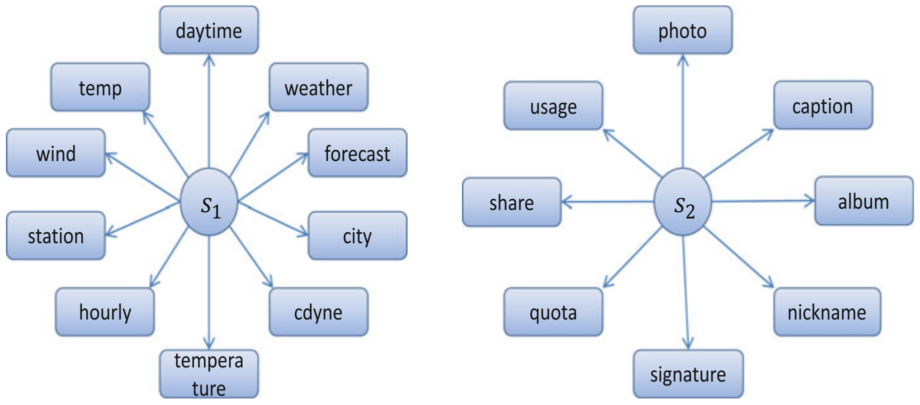


**Fig. 3** Content words of two real Web services

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 [4,5]. 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 four steps, we can obtain the final content vector. Figure 3 shows the content vectors of two real Web services where $s_1$ is a weather forecast service,[7] and $s_2$ is a photograph sharing service.[8] Their content vectors are quite relevant to their function and have limited elements only, i.e., 10 and 8, respectively. Through our observation, the dimension of content vector of most Web services for experiments (i.e., 15,969 real web service) is in the range of 10–30.

In this paper, we use Normalized Google Distance (NGD) [34] to compute the content-level similarity between two Web services. Given Web services $s_1$ and $s_2$, and their content vector $con_{s_1}$ and $con_{s_2}$, their content-level similarity is:

$$Sim_{con}(s_1, s_2) = \frac{\sum_{w_i \in con_{s_1}} \sum_{w_j \in con_{s_2}} sim(w_i, w_j)}{|con_{s_1}| \times |cons_2|}, \tag{3}$$

where $|con_{s_1}|$ means the dimension of content vector $con_{s_1}$, and the equation for computing the similarity between two words is defined as:

$$sim(w_i, w_j) = 1 - NGD(w_i, w_j) \tag{4}$$

In (4), we compute the similarity between two words using NGD based on the word coexistence in the Web pages. As Rudi et al. [34] proposed, the NGD between two terms $w_i$ and $w_j$ is as follows:

[7] http://ws.cdyne.com/WeatherWS/Weather.asmx?wsdl.

[8] http://gordan.gorodok.net/sharpcast.wsdl.

$$NGD(w_i, w_j) = \frac{max\{\log f(w_i), \log f(w_j)\} - \log f(w_i, w_j)}{\log N - min\{\log f(w_i), \log f(w_j)\}}, \tag{5}$$

where $f(w_i)$ denotes the number of pages containing $w_i$, and $f(w_i, w_j)$ denotes the number of pages containing both $w_i$ and $w_j$, as reported by Google. $N$ is the total number of Web pages searched by Google.

#### *4.1.2 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 a 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. 4 shows, the type of element *ProcessForm* (which we name $type_1$) is a complextype that 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 two string-type parameters and three 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 as follows

$$Sim_{type}(s_1, s_2) = \frac{2 \times Match(Type_{s_1}, Type_{s_2})}{|Type_{s_1}| + |Type_{s_2}|}, \tag{6}$$

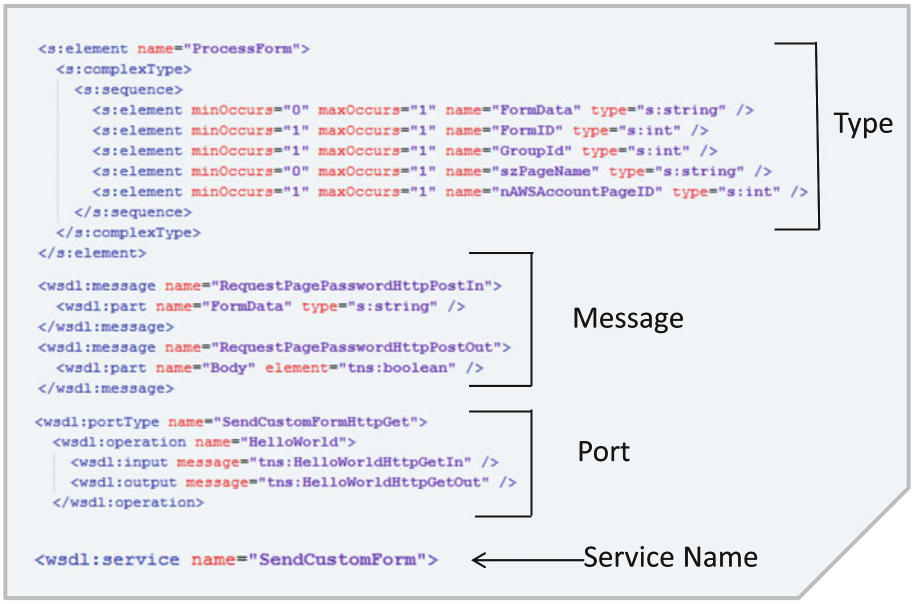
where $Type_{s_1}$ means the set of defined types in the WSDL document of $s_1$, $Match(Type_{s_1}, Type_{s_2})$ means the number of matched types between these two services, and $|Type_{s_1}|$ is the cardinality of $Type_{s_1}$.

#### *4.1.3 Message*

Message is used to deliver parameters between different operations. One message contains one or more parameters, and each parameter is associated with a type as we discussed above. Message definition is typically considered as an abstract definition of the message content which defines the name and type of the parameter contained in the message. Figure 4 shows two simple message definitions. In the first definition, the message named as *Request-PagePasswordHttpPostIn* contains one parameter *FormData* of a *string* type. In the second definition, *RequestPagePasswordPostOut* contains one parameter *Body* of a complextype named as *tns:boolean*. Similar to Eq. (6), we match the messages' structures to compute the message-level similarity between Web services.

#### *4.1.4 Port*

The portType element combines multiple message elements to form a complete one-way or round-trip operation. Figure 4 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 Eq. (6) to compute the port-level similarity.



```
<s:element name="ProcessForm">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="FormData" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="FormID" type="s:int" />
      <s:element minOccurs="1" maxOccurs="1" name="GroupId" type="s:int" />
      <s:element minOccurs="0" maxOccurs="1" name="szPageName" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="nAWSAccountPageID" type="s:int" />
    </s:sequence>
  </s:complexType>
</s:element>

<wsdl:message name="RequestPagePasswordHttpPostIn">
  <wsdl:part name="FormData" type="s:string" />
</wsdl:message>
<wsdl:message name="RequestPagePasswordHttpPostOut">
  <wsdl:part name="Body" element="tns:boolean" />
</wsdl:message>

<wsdl:portType name="SendCustomFormHttpGet">
  <wsdl:operation name="HelloWorld">
    <wsdl:input message="tns:HelloWorldHttpGetIn" />
    <wsdl:output message="tns:HelloWorldHttpGetOut" />
  </wsdl:operation>

<wsdl:service name="SendCustomForm">
```

**Fig. 4** Types, message, port, and service name in a WSDL document

### *4.1.5 Service name*

As the service name (*sname*) can at least partially reflect the service function, it is also 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. 4 can be separated into three words: *Send*, *Custom*, and *Form*. A simple version of word segmentation is to split the service name according to the capital letters. However, its performance is not satisfied due to different naming conventions. In this paper, we first use the capital letters to split the service name and then manually adjust the final result. After the process of word segmentation, $SName_{s_1}$, the name of service $s_1$, can be presented as a set of words. Then, we can use Eq. (3) to compute the *sname*-level similarity between Web services.

### *4.1.6 Tag*

The tagging data of Web services describe their function 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. Similar to above five features extracted from the WSDL document, tagging data are also treated as a feature in the process of similarity computation.

Given a Web service $s_i$ with 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 [35] method, we can calculate the tag-level similarity between two Web services $s_i$ and $s_j$ as follows:

$$Sim_{tag}(s_i, s_j) = \frac{|T_i \cap T_j|}{|T_i \cup T_j|}, \tag{7}$$

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., $|T_i \cup T_j| = |T_i| + |T_j| - |T_i \cap T_j|$.

However, the real tag-level similarity between Web services cannot be reflected if we directly use Eq. (7) to compute the similarity based on the original tagging data. Some inherent properties of Web service tagging data, e.g., uneven tag distribution and noisy tags, impact the reliability of tag-level similarity. In Sect. 5, a Web service tag recommendation strategy will be introduced to handle this problem.

### 4.2 Similarity integration and service clustering

In the above subsection, we calculate feature-level similarity and tag-level similarity among Web services, i.e., $Sim_{con}$, $Sim_{type}$, $Sim_{mes}$, $Sim_{port}$, $Sim_{sname}$, and $Sim_{tag}$. In this section, we integrate them into a global similarity for the purpose of Web service clustering.

Before similarity integration, it should be noted that the WSDL document is created by service providers and tagging data reflect users' knowledge. Thus, these two kinds of similarities should be treated separately. In this paper, we first merge the similarities of features extracted from WSDL documents to derive a WSDL-level similarity and then integrate WSDL-level similarity and tag-level similarity into a global one.

We measure the WSDL-level similarity between Web services $s_i$ and $s_j$ as follows:

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

where $w_1, \ldots, w_5$ are the user-defined weights of *Content, Type, Message, Port*, and *Service Name*, respectively, with their sum being unity. The global similarity $GSim(s_i, s_j)$ between $s_i$ and $s_j$ is defined as follows:

$$GSim(s_i, s_j) = (1 - \lambda) Sim_{wsdl}(s_i, s_j) + \lambda Sim_{tag}(s_i, s_j), \tag{9}$$

where $\lambda \in [0, 1]$ is the weight of the tag-level similarity for the purpose of balancing. When $\lambda \in (0, 1)$, $GSim(s_i, s_j)$ is equal to 1 if the WSDL documents and tags of these two services are identical, and $GSim(s_i, s_j)$ is equal to 0 if both the WSDL documents and the tags of these two services are completely different. Specifically, only the WSDL documents are utilized for Web service clustering when $\lambda = 0$, while we cluster Web services according to the tag-level similarity when $\lambda = 1$.

As for Web service clustering, we propose to use K-means [36] clustering approach. K-means is a widely adopted clustering algorithm that is simple and fast. Its drawback is that the number of clusters has to be predefined manually before clustering.

## 5 WSTRec

Some inherent properties of Web service tagging data, e.g., uneven tag distribution and noisy tags, impact the reliability of tag-level similarity. In this section, we introduce a novel Web service tag recommendation strategy called *WSTRec*, in which tag mining, tag occurrence, and semantic relevance are all adopted.

## 5.1 Tag mining

Through our observation, many Web services including new members do not have tags. In this case, we have to mine some tags from the textual features of the services first and then use a tag recommendation strategy to improve the quality of the tagging data.

The assumption of tag mining is that the more important a term in the textual features, the more probable for this term to be attached as a tag. In this paper, we use a well-accepted method *Term Frequency-Inverse Document Frequency* (*TF-IDF*) [33,37], which is widely adopted in the domain of information retrieval and nature language processing. Specifically, textual features used for tag mining contain three parts: (1) the service name, (2) the service description given by service provider, and (3) content words extracted from the WSDL document.

Given a Web service $s_1$ and a term $t$ extracted from $s_1$'s textual features, *TF-IDF* is calculated as follows:

$$\textit{TF-IDF}_{t,s_1} = TF_{t,s_1} \times IDF_t, \tag{10}$$

where $TF_{t,s_1}$ denotes the term frequency (i.e., number of occurrences) of term $t$ in $s_1$, and $IDF_t$ is the inverse document frequency defined as follows:

$$IDF_t = \log \frac{N}{df_t}, \tag{11}$$

where $N$ is the number of services in the collection, and $df_t$ denotes the document frequency of the term $t$, that is, the number of services that contain the term $t$. As the term with higher value of *TF-IDF* is more important, we can choose the top terms as the initial tags.

## 5.2 Tag co-occurrence

Typically, an initial set of tags $\mathcal{I}_s$ associated with a Web service $s$ is provided to the recommendation method, which outputs a set of related tags $\mathcal{C}_s$, where $\mathcal{I}_s \bigcap \mathcal{C}_s = \emptyset$. Tag co-occurrence is a commonly used method for tag recommendation.

In our preliminary work, we propose a simple tag co-occurrence-based approach by employing Jaccard coefficient method [35]. In this paper, we extract association rules to reflect the tag co-occurrence and compute the scores of candidate tags according to the association rules.

### *5.2.1 Association rule extraction*

In the process of tag recommendation, we separate Web services into two parts: (1) training data (denoted as $\mathcal{T}$), which have many associated tags; (2) testing data (denoted as $\mathcal{P}$), which have few tags and need tag recommendation. Given a Web service $s_t$ in training data, $s_t$ is presented as $< \mathcal{I}_{s_t}, \mathcal{Y}_{s_t} >$, where $\mathcal{I}_{s_t}$ contains all tags associated with service $s_t$, while $\mathcal{Y}_{s_t}$ is empty. As for a Web service $s_p$ in the testing data, $s_p =< \mathcal{I}_{s_p}, \mathcal{Y}_{s_p} >$, where $\mathcal{I}_{s_p}$ is the initial tag sets, and $\mathcal{Y}_{s_p}$ are recommended tags based on tags in $\mathcal{I}_{s_p}$ according to the association rules extracted from the training data $\mathcal{T}$.

Table 2 shows a simple scenario of tag recommendation. In this scenario, we have to recommend some tags to $s_{p1}$ based on $\mathcal{I}_{s_{p1}}$ (i.e., report and ZIP) according to the association rules extracted from $\mathcal{T}$. The definition of an association rule is as follows.

**Table 2** Examples of training data and testing data

| | $\mathcal{I}$ | $\mathcal{Y}$ |
|---|---|---|
| $\mathcal{T}$ | | |
| $s_{t1}$ | Weather ZIP report USA | $\emptyset$ |
| $s_{t2}$ | USA ZIP company free | $\emptyset$ |
| $s_{t3}$ | Weather station report signup | $\emptyset$ |
| $s_{t4}$ | City station ZIP temperature | $\emptyset$ |
| $\mathcal{P}$ | | |
| $s_{p1}$ | Report ZIP | ? |

**Definition 1** An association rule is denoted as $\mathcal{X} \xrightarrow{\theta} y$, where the antecedent $\mathcal{X}$ is a set of tags, and the consequent $y$ is a predicted tag. The size of association rule $\mathcal{X} \xrightarrow{\theta} y$ is the number of tags in $\mathcal{X}$, that is, $|\mathcal{X}|$. The strength of the association between antecedent $\mathcal{X}$ and consequent $y$ is given by $\theta$, which is the conditional probability of $y$ being in $\mathcal{Y}_{s_p}$ given that $\mathcal{X} \subseteq \mathcal{I}_{s_p}$.

Take the tagging data of $s_{t1}$ as an example, we can extract many association rules from these four tags according to Definition 1, e.g., $weather \longrightarrow report$, $(weather, ZIP) \longrightarrow report$, and $(report, USA) \longrightarrow weather$. Given $n$ tags, the number of rules $NR(n)$ extracted from these tags is:

$$NR(n) = \sum_{m=1}^{n-1} C_n^m \times (n-m)$$
$$= n \sum_{m=1}^{n-1} C_{n-1}^m = n(2^{n-1} - 1) \tag{12}$$

According to Eq. (10), it can be discovered that the association rule space is very large. To reduce it, we propose the following lemma that could be used to control the extraction of association rules.

**Lemma 1** *Rule $\mathcal{X} \xrightarrow{\theta} y$ is a useful association rule for testing service $s_p$, if and only if $\mathcal{X} \subseteq \mathcal{I}_{s_p}$.*

Take the rule $(report, USA) \longrightarrow weather$ and $s_{p1}$ in Table 2 as an example, this rule is useless as the set of initial tags of $s_{p1}$ does not contain $USA$. Thus, the extraction of association rules depends on the initial tags in testing data. We only extract rules when the antecedent $\mathcal{X}$ is the subset of or equal to the initial tags of testing service. For illustrative purposes, we create a projected training data, $\mathcal{T}^{\mathcal{P}}$, which consists of services $s_t^p =< \mathcal{I}_{s_t}^{s_p}, \mathcal{Y}_{s_t}^{s_p} >$, where $\mathcal{I}_{s_t}^{s_p} = \{\mathcal{I}_{s_t} \bigcap \mathcal{I}_{s_p}\}$ and $\mathcal{Y}_{s_t}^{s_p} = \{\mathcal{I}_{s_t} - \mathcal{I}_{s_t}^{s_p}\}$. Table 3 shows the projected training data based on data in Table 2.

For any service $s_t^p$ in the projected training data, $\mathcal{I}_{s_t}^{s_p} \subseteq \mathcal{I}_{s_p}$, we can extract association rules in the following sense: $\mathcal{X} \xrightarrow{\theta} y$, $\mathcal{X} \subseteq \mathcal{I}_{s_t}^{s_p}$, and $y \in \mathcal{Y}_{s_t}^{s_p}$. According to Lemma 1, 13 association rules could be extracted from Table 3 and some example rules are as follows:

1. $ZIP \xrightarrow{\theta=0.67} USA$
2. $report \xrightarrow{\theta=0.5} station$
3. $report, ZIP \xrightarrow{\theta=1} weather$

**Table 3** Projected training data for rule extraction

| $\mathcal{T}^{\mathcal{P}}$ | $\mathcal{I}^{\mathcal{P}}$ | $\mathcal{Y}^{\mathcal{P}}$ |
|---|---|---|
| $s_{t1}^{p1}$ | Report ZIP | Weather USA |
| $s_{t2}^{p1}$ | ZIP | USA company free |
| $s_{t3}^{p1}$ | Report | Weather station signup |
| $s_{t4}^{p1}$ | ZIP | City station temperature |

### 5.2.2 *Rule-based scoring*

In order to recommend tags that are more likely to be associated with service $s_p \in \mathcal{P}$, we have to give a score to each candidate tag by utilizing rules in $R_{s_p}$ which is the set of association rules extracted based on $s_p$ and $\mathcal{T}$. Given a candidate tag $y$, the rule-based scoring function is as follows:

$$\Phi(s_p, y) = \sum_{\mathcal{X} \subseteq \mathcal{I}_{s_p}} \theta(\mathcal{X} \longrightarrow y) \tag{13}$$

## 5.3 Semantic relevance

Tags recommended only based on tag co-occurrence sometimes may be irrelevant to the target Web service. Take the rule $(report, ZIP) \overset{\theta=1}{\longrightarrow} weather$ extracted from Table 3 as an example. Tag $weahter$ maybe wrongly recommended to a document sharing service which has tag $report$ as the strong association rule between $weather$ and $report$. Thus, the relevance between a candidate tag and the target Web service should be considered in the process of tag recommendation. Further, the introduction of relevance metric can also filter out the noisy tags (e.g., misspellings or unrelated terms) or reduce their importance.

Given a candidate tag $y$ and the target Web service $s_p$, the semantic relevance between $s_p$ and $y$, denoted as $\Omega(s_p, y)$, is generated by summing the semantic similarity between tag $y$ and textual features of $s_p$, i.e., service name, service description, and content vector. In particular, the semantic similarity between $y$ and a textual feature $\mathcal{F}_{s_p}^i$ is calculated as follows:

$$Sem(\mathcal{F}_{s_p}^i, y) = \begin{cases} 1, & y \in \mathcal{F}_{s_p}^i \\ \max_{t \in \mathcal{F}_{s_p}^i} (1 - NGD(y, t)), & y \notin \mathcal{F}_{s_p}^i \end{cases} \tag{14}$$

$\Omega(s_p, y)$ is calculated as follows:

$$\Omega(s_p, y) = \sum_{i=1}^{N} Sem(\mathcal{F}_{s_p}^i, y), \tag{15}$$

where $N$ is the number of textual features, which is 3 in this paper.

## 5.4 Tag ranking

$\Phi(s_p, y)$ reflects the association rules between candidate tag $y$ and other tags, while $\Omega(s_p, y)$ shows the semantic relevance between $y$ and $s_p$. In *WSTRec*, we combine these two measurements to rank candidate tags for the purpose of recommendation. Given $s_p$, the final score of candidate tag $y$ is calculated as follows:

$$Score(s_p, y) = (1 - \delta)\Phi(s_p, y) + \delta\Omega(S_p, y), \tag{16}$$

where $\delta$ is a weighting factor to balance the importance of tag co-occurrence and semantic relevance in the process of tag recommendation.

## 6 Experiment

In this section, we present an experimental evaluation of the proposed approach by measuring: (1) performance of different Web service clustering approaches in terms of precision and recall; (2) impact of $\mathcal{N}$ (Number of Web services in one cluster), $\mathcal{C}$ (Number of clusters), $\lambda$, and $\delta$ to the performance of Web service clustering; (3) performance of *WSTRec*; and (4) impact of $\delta$ to the performance of *WSTRec*.

### 6.1 Experimental setup

To evaluate the performance of Web service clustering approaches and tag recommendation strategies, we crawl 15,968 real Web services [9] from the Web service search engine Seekda! For each Web service, we get the data of service name, service description given by service providers, WSDL document, tags, availability information, and the name of the service providers.

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

### 6.2 Performance of Web service clustering

To create the ground truth for Web service clustering performance evaluation, we randomly select 320 Web services from the crawled 15,968 Web service. Specifically, there are 50 Web services with no tags, and a tag mining method has to be employed to mine some initial tags. We perform a manual classification of these 320 Web services to serve as the ground truth for the clustering approaches. Specifically, we distinguish the following four function categories: "Weather," "Email," "Stock," and "Education." There are 42 Web services in "Weather" category, 37 in "Communication," 39 in "Stock," and 52 in "Finance." Due to the space limitation, the detailed information is omitted here.

To evaluate the performance of Web service clustering, we introduce two metrics (precision and recall) which are widely adopted in the information retrieval domain.

$$\begin{aligned} Precision_{c_i} &= \frac{succ(c_i)}{succ(c_i) + mispl(c_i)} \\ Recall_{c_i} &= \frac{succ(c_i)}{succ(c_i) + missed(c_i)}, \end{aligned} \tag{17}$$

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 $c_i$, and $missed(c_i)$ is the number of services that should be placed into $c_i$ but are placed into another.

[9] STag Dataset 1.0, collected by Liang Chen (cliang@zju.edu.cn) and Johnny Jian (johnnyjian@gmail.com).

In this section, we compare the performance 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 Eq. (8)). This approach was adopted in some related work [4,5].
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 Eq. (9). Specifically, *WSTRec* is not employed in $WTCluster^1$.
3. ***WTCluster***$^2$. In this approach, all the processes in our proposed Web service clustering architecture, i.e., Fig. 2, are adopted. Web service clustering is performed after the execution of *WSTRec*.

Figure 5 shows the comparison results. For simplicity, we set $w_i = 0.2$, $i \in \{1, 2, 3, 4, 5\}$ in all the following experiments. The values of other related parameters are set as $\lambda = 0.5$ and $\delta = 0.25$. 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, the tags of Web services contain much information, such as service function, location, and other semantical information. Utilizing the tag information improves the performance of Web service clustering.

Further, it can be observed that the approach $WTCluster^2$ which contains the process of tag recommendation outperforms $WTCluster^1$ approach. The introduction of *WSTRec* produces gains in precision of up to 14.1 % and generates improvements in recall of up to 10.8 %. This is because the implementation of *WSTRec* recommends some related tags to the Web services with few tags, which enriches the information of Web services.

## 6.3 Impact of $\mathcal{C}$ on the performance of clustering

It should be noted that if the functions of Web services in the same cluster differ a lot, litter help can be provided by Web service clustering to the discovery of Web service. The best situation of Web service clustering is that functions of all Web services in the same cluster are single, such as the cluster of $weatherforecast$. Thus, the number of service clusters will be a large number in the real application, for the purpose of facilitating Web service discovery.

In the experiments leading to the results in Fig. 5, the number of Web service clusters is $\mathcal{C} = 4$, which is too small compared with the number of function categories in real applications. In this section, we evaluate the performance of Web service clustering as $\mathcal{C}$ varies. Besides the

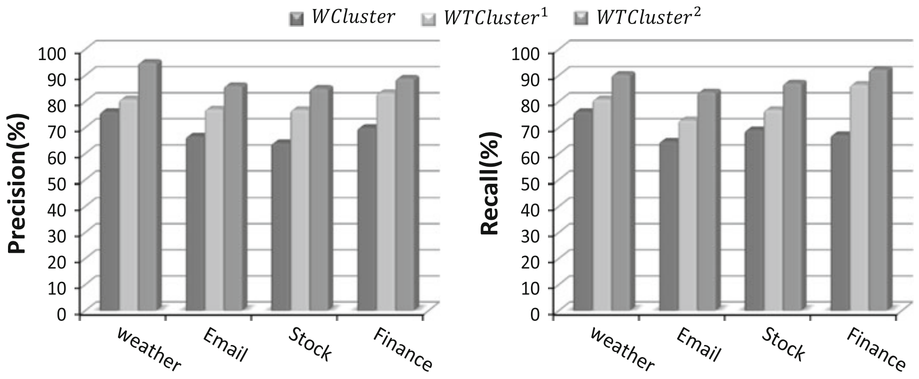


**Fig. 5** Comparison of Web service clustering performance

Web services used in the prior experiments, we add many other Web services with different function categories, e.g., *Tourist, University, and HR*. For each addition, there are about 30–40 Web services in it. The evaluation metrics for this experiment are *Avg-precision* and *Avg-recall*, which are the average values of all clusters' precisions and recalls.

$$\begin{aligned} Avg-precision &= \frac{1}{\mathcal{C}}\sum_{i=1}^{\mathcal{C}} precision_{c_i} \\ Avg-recall &= \frac{1}{\mathcal{C}}\sum_{i=1}^{\mathcal{C}} recall_{c_i} \end{aligned} \tag{18}$$

Figure 6 shows the performance of Web service clustering versus $\mathcal{C}$ where $\lambda = 0.5$ and $\delta = 0.25$. *Avg-precision* and *Avg-recall* of $WCluster$ decrease when $\mathcal{C}$ increases from 4 to 12. Especially, the performance of $WCluster$ decreases up to 10.2 % when $\mathcal{C} = 12$. Compared with $WCluster$, the performance of $WTCluster^1$ and $WTCluster^2$ is quiet stable. This is because the introduction of tagging data improves the distinguishability among Web services belonging to different function category. Thus, we can draw the conclusion that the scalability of our proposed Web service clustering approaches is good in terms of $\mathcal{C}$.

## 6.4 Impact of $\mathcal{N}$ on the performance of clustering

With the development of service-oriented computing, more and more Web services with similar functions but different QoS appear in the Internet, that is, the number of Web services in one function category may be large in the very near future. In the above experiments, the number of Web services in each cluster is about 40, which is relatively small. In this part, we evaluate the performance of our proposed clustering approaches when the number of services in each cluster (i.e., $\mathcal{N}$) is large. We set $\lambda = 0.5$, $\delta = 0.25$, and $\mathcal{C} = 4$.

Figure 6c, d shows the performance of three clustering approaches with $\mathcal{N}$. Their precision performance results are different when $\mathcal{N}$ increases: the precision values of $WTCluster^1$ and $WTCluster^2$ first decrease and then increase, while the trend of $WCluster$ approach is the opposite. As for the trend of the recall values, they increase with $\mathcal{N}$, that is, with the increase in the number of Web services in a cluster, the performance of Web service clustering will be better. This is because the increase in Web services enriches the corpus and provides more association rules.

## 6.5 Impact of $\lambda$

In our proposed Web service clustering approach, both WSDL document and tags are utilized with parameter $\lambda$ to balance the weights of two data in the calculation of global similarity

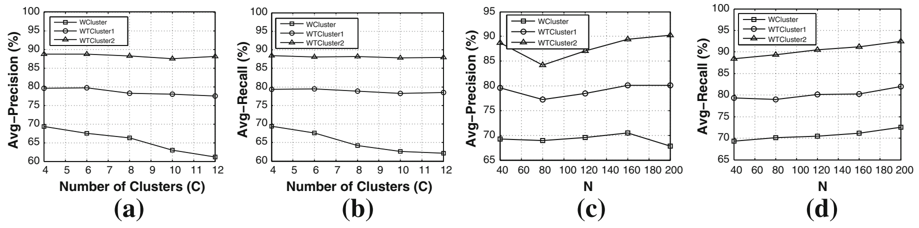


**Fig. 6** Impact of $\mathcal{C}$ and $\mathcal{N}$

between Web services. When $\lambda = 0$, $WTCluster$ equals to $WCluster$. When $\lambda = 1$, only tagging data are utilized for clustering. In this part, we try to find an optimal value of $\lambda$ which leads to the best performance of service clustering. Thus, we need to evaluate the performance of $WTCluster^2$ approach only. Specifically, set other parameters as follows: $\mathcal{C} = 4, \mathcal{N} = 40, \delta = 0.25$.

Figure 7a, b shows the performance of service clustering versus $\lambda$. From Fig. 7a, we can find the optimal value of $\lambda$ shifts from 0.3 to 0.5 when the best performance of precision is reached. The optimal $\lambda$ shifts from 0.6 to 0.8 when the value of recall reaches its highest point. Thus, no optimal value of $\lambda$ can make $WTCluster^2$ reach its best performance in terms of both precision and recall. In this paper, therefore, we set $\lambda$ as 0.5 to include the trade-off.

## 6.6 Impact of $\delta$

Tag recommendation result is determined by the scoring function which makes use of tag co-occurrence and semantic relevance. $\delta$ is employed to balance the weights of tag co-occurrence and semantic relevance in the final score. As tag-based similarity plays an important role in the calculation of global similarity, the value of $\delta$ indirectly impacts the performance of Web service clustering.

Figure 7c, d shows the trend of clustering performance versus $\delta$. From them, we can find that the value of precision and recall first increases and then decreases when the value of $\delta$ varies from 0 to 0.5. Further, when $\delta$ shifts from 0.2 to 0.3, both precision and recall reach their highest points. Thus, we choose 0.25 as the optimal value of $\delta$ in this paper.

## 6.7 Evaluation of tag recommendation strategies

Before evaluating the performance of $WSTRec$, we select 2,200 Web services, which contain 1,254 unique tags, as the dataset for evaluation. Specifically, there are 400 Web services with no initial tags in these 2,200 services. The ground truth is manually created through a blind review pooling method, where for each of the 2,200 Web services, the top five recommendations from *WSTRec* were taken to construct the pool. The volunteers (i.e., some master and undergraduate students in the laboratory) were then asked to evaluate the descriptiveness of each of the recommended tags in the context of the Web services. We provide the WSDL documents and Web service descriptions to the volunteers for helping them. The volunteers were asked to judge the descriptiveness on a three-point scale: *very good*, *good*, and *not good*. The distinction between *very good* and *good* is defined to make the assessment task conceptually easier for the users. Finally, we get 1,948 *very good* judgements (19.6 %), 2,145 *good* judgements (21.6 %), and 5,854 *not good* judgements (58.8 %).

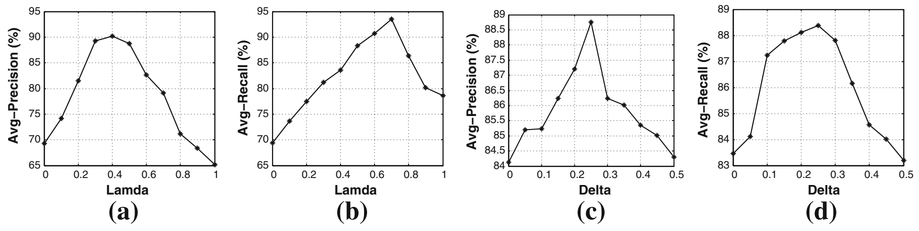


**Fig. 7** Impact of $\lambda$ and $\delta$

To evaluate the performance of *WSTRec*, we adopt two widely adopted 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 taken 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.

In our prior work [31], we propose two simple tag recommendation strategies, i.e., *Sum* and *Vote*. In this section, we compare the performance of *WSTRec* with *Sum* and *Vote*.

Table 4 shows the S@K performance of three tag recommendation strategies, where $\mathcal{G}$ means the number of initial tags which the target Web service contains. Specifically, if $s_i$ does not have initial tags, tag mining technique is implemented to mine $\mathcal{G}$ relevant tags as the initial tags. Take *WSTRec* as an example; when $\mathcal{G}$ varies from 1 to 2, the average value of S@K is over 0.77, i.e., more than 77 % recommended tags obtain *good* or *very good* descriptiveness. For all three strategies, it can be observed that when $\mathcal{G}$ increases, the value of S@K increases. When K increases, S@K decreases in most cases. However, in any situation, *WSTRec* outperforms the other two strategies in terms of S@K.

Table 5 shows the comparison of three tag recommendation strategies in terms of P@K. From Table 5, it can be observed that P@K decreases when $\mathcal{G}$ increases. This is because the number of relevant tags to a certain Web service is limited. When $\mathcal{G}$ increases, the number of relevant tags left decreases, which lowers P@K. In addition, P@K achieves its largest value when $K = 1$ and decreases when K increases in most cases. Similar to the observation in Table 4, *WSTRec* outperforms the other two recommendation strategies in terms of P@K.

## 7 Conclusion

In this paper, we propose to utilize tagging data to improve the performance of traditional WSDL document-based Web service clustering for the purpose of more accurate Web service discovery. In our proposed clustering approach, Web services are clustered according to the

**Table 4** S@K comparison of three tag recommendation strategies

| $\mathcal{G}$ | Method | $K = 1$ | $K = 2$ | $K = 3$ | $K = 4$ | $K = 5$ |
|---|---|---|---|---|---|---|
| 1–2 | *Sum* | 0.7584 | 0.7387 | 0.6925 | 0.6901 | 0.6847 |
| | *Vote* | 0.6913 | 0.6548 | 0.6987 | 0.7125 | 0.7048 |
| | *WSTRec* | **0.8102** | **0.7843** | **0.7776** | **0.7489** | **0.7463** |
| 3–5 | *Sum* | 0.7665 | 0.7444 | 0.7281 | 0.6998 | 0.6914 |
| | *Vote* | 0.7485 | 0.7254 | 0.7351 | 0.7041 | 0.7015 |
| | *WSTRec* | **0.8351** | **0.8025** | **0.7877** | **0.7654** | **0.7414** |
| >5 | *Sum* | 0.7759 | 0.7744 | 0.7568 | 0.7358 | 0.7025 |
| | *Vote* | 0.7895 | 0.7768 | 0.7712 | 0.7489 | 0.7345 |
| | *WSTRec* | **0.8598** | **0.8248** | **0.8079** | **0.7758** | **0.7681** |

Larger values are shown in bold

**Table 5** P@K comparison of three tag recommendation strategies

| $\mathcal{G}$ | Method | $K = 1$ | $K = 2$ | $K = 3$ | $K = 4$ | $K = 5$ |
|---|---|---|---|---|---|---|
| 1–2 | *Sum* | 0.6548 | 0.6143 | 0.5989 | 0.5418 | 0.4986 |
| | *Vote* | 0.6859 | 0.6458 | 0.6215 | 0.6249 | 0.5615 |
| | *WSTRec* | **0.7102** | **0.7015** | **0.6899** | **0.6874** | **0.6589** |
| 3–5 | *Sum* | 0.6672 | 0.6415 | 0.6015 | 0.5687 | 0.5145 |
| | *Vote* | 0.6245 | 0.6325 | 0.6018 | 0.6245 | 0.6178 |
| | *WSTRec* | **0.7018** | **0.6948** | **0.6812** | **0.6741** | **0.6254** |
| >5 | *Sum* | 0.5658 | 0.5147 | 0.4958 | 0.4256 | 0.3958 |
| | *Vote* | 0.6214 | 0.5847 | 0.5141 | 0.4478 | 0.3845 |
| | *WSTRec* | **0.6544** | **0.6014** | **0.5663** | **0.4979** | **0.4875** |

Larger values are shown in bold

global similarities among services, which are calculated based on tagging data and five features extracted from the WSDL document.

Furthermore, we propose a hybrid Web service tag recommendation approach *WSTRec* to improve the performance limited by the uneven tag distribution and noisy tags. Tag co-occurrence, tag mining, and semantic relevance are adopted in *WSTRec*. Specifically, tag mining technique is employed to mine some initial tags for Web services with no tags.

To evaluate the performance of Web service clustering, we crawl 15,968 real Web services from the Web service search engine Seekda! Compared with traditional WSDL-based Web service clustering, our proposed clustering approach produces gains in both precision and recall of up to 14 % in most cases. Moreover, *WSTRec* outperforms two widely accepted tag recommendation strategies in terms of both S@K and P@K.

The scale of tag-related experiments in this paper is small, due to the limited tagging data of Web services. In our future work, we plan to enlarge the scale of Web service tagging data by attracting more users to annotate Web services in our proposed Titan[10] Web service search engine. Tagging data of Web services will also be employed in other applications, e.g., Web service search and collaborative filtering-based Web service recommendation. Furthermore, we will conduct more research in utilizing social network information to facilitate Web service mining.

**Acknowledgments** This research was partially supported by the National Technology Support Program under the grant of 2011BAH16B04, the National Natural Science Foundation of China under the grant of No. 61173176, Science and Technology Program of Zhejiang Province under the grant of 2008C03007, National High-Tech Research and Development Plan of China under the Grant No. 2011AA010501, and a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK 415409).

## References

1. Al-Masri E, Mahmoud QH (2008) Investigating web services on the world wide web. In: Proceedings of international World Wide Web conference, pp 795–804
2. Nayak, Richi (2008) Data mining in web service discovery and monitoring. Int J Web Serv Res 5(1):62–80
3. Lim SY, Song MH, Lee SJ (2004) The construction of domain ontology and its application to document retrieval. Lect Notes Comput Sci 3261:117–127

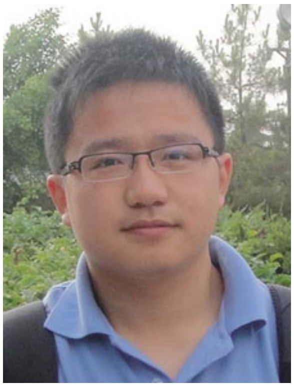
[10] http://ccnt.zju.edu.cn:8080.

4. Elgazzar K, Hassan AE, Martin P (2009) Clustering wsdl documents to bootstrap the discovery of web services. In: Proceedings of international conference on Web services, pp 147–154
5. Liu W, Wong W (2009) Web service clustering using text mining techniques. Int J Agent-Oriented Softw Eng 3(1):6–26
6. Lipczak M, Hu Y, Kollet Y, Milios E (2009) Tag sources for recommendation in collaborative tagging systems. ECML PKDD Discov Chall 497:157–172
7. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of international conference on very large data bases, pp 487–499
8. Wu Z, Deng S, Li Y, Wu J (2009) Computing compatibility in dynamic service composition. Int J Knowl Inf Syst 19(1):107–129
9. Hu C, Zhu Y, Huai J, Liu Y, Ni LM (2007) S-club: an overlay-based efficient service discovery mechanism in crown grid. Int J Knowl Inf Syst 12(1):55–75
10. Liang QA, Chung J-Y, Miller S (2007) Modeling semantics in composite web service requests by utility elicitation. Int J Knowl Inf Syst 13(3):367–394
11. Wu J, Chen L, Xie Y, Zheng Z (2012) Titan: a system for effective web service discovery. In: Proceedings of international conference companion on World Wide Web, pp 441–444
12. Agarwal S, Studer R (2006) Automatic matchmaking of web services. In: Proceedings of international conference on Web services, pp 45–54
13. Klusch M, Fries B, Sycara K (2006) Automated semantic web service discovery with owls-mx. In: Proceedings of international conference on autonomous agents and multiagent systems, pp 915–922
14. Benatallah B, Hacid M, Leger A, Rey C, Toumani F (2005) On automating web services discovery. Int J Very Large Data Bases 14(1):84–96
15. Zhang Y, Zheng Z, Lyu MR (2010) Wsexpress: a qos-aware search engine for web services. In: Proceedings of international conference on Web services, pp 91–98
16. Dong X, Halevy A, Madhavan J, Nemes E, Zhang J (2004) Similarity search for web services. In: Proceedings of international conference on very large data bases, pp 372–383
17. Hu S, Muthusamy V, Li G, Jacobsen HA (2008) Distributed automatic service composition in large-scale systems. In: Proceedings of distributed event-based systems conference, pp 233–244
18. Liu F, Shi Y, Yu J, Wang T, Wu J (2010) Measuring similarity of web services based on wsdl. In: Proceedings of international conference on Web services, pp 155–162
19. Ran S (2003) A model for web services discovery with qos. ACM Sigecom Exch 4(1):1–10
20. Lara R, Corella MA, Castells P (2006) A flexible model for web service discovery. In: Proceedings of international conference on very large data bases
21. Zheng Z, Ma H, Lyu MR, King I (2009) Wsrec: a collaborative filtering based web service recommender system. In: Proceedings of international conference on Web services, pp 437–444
22. Zheng Z, Ma H, Lyu MR, King I (2011) Qos-aware web service recommendation by collaborative filtering. IEEE Trans Serv Comput 4(2):140–152
23. Wu J, Chen L, Feng Y, Zheng Z, Zhou M, Wu Z (2013) Predicting quality of service for selection by neighborhood-based collaborative filtering. IEEE Trans Syst Man Cybern Part A 43(2):428–439
24. Hao Y, Junliang C, Xiangwu M, Bingyu Q (2007) Dynamically traveling web service clustering based on spatial and temporal aspects. Lect Notes Comput Sci 4802:348–357
25. Platzer C, Rosenberg F, Dustdar S (2009) Web service clustering using multidimensional angles as proximity measures. ACM Trans Intern Technol 9(3):1–26
26. Bianchini D, Antonellis VD, Pernici B, Plebani P (2006) Ontology-based methodology for e-service discovery. ACM J Inf Syst 31(4):361–380
27. Sun P, Jiang C (2008) Using service clustering to facilitate process-oriented semantic web service discovery. Chin J Comput 31(8):1340–1353
28. Pop CB, Chifu VR, Salomie I, Dinsoreanu M, David T, Acretoaie V (2010) Semantic web service clustering for efficient discovery using an ant-based method. Stud Comput Intell 315:23–33
29. Dasgupta S, Bhat S, Lee Y (2010) Taxonomic clustering of web service for efficient discovery. In: Proceedings of international conference on information and, knowledge management, pp 1617–1620
30. WeiLiu, Wong W (2008) Discovering homogeneous service communities through web service clustering. Serv Oriented Comput Agents Semant Eng 5006:69–82
31. Chen L, Hu L, Wu J, Zheng Z, Ying J, Li Y, Deng S (2011) Wtcluster: utilizing tags for web service clustering. In: Proceedings of international conference on service oriented, computing, pp 204–218
32. Porter MF (1980) An algorithm for suffix stripping. Program 14(3):130–137
33. Church K, Gale WA (1995) Inverse document frequency (idf): a measure of deviations from poisson. In: Proceedings of the ACL 3rd workshop on very large corpora, pp 121–130
34. Cilibrasi RL, Vitnyi PMB (2007) The google similarity distance. IEEE Trans Knowl Data Eng 19(3):370–383

35. Jain A, Dubes R (1988) Algorithms for clustering data. Prentice Hall, New Jersey
36. MacQueen JB (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth symposium on math, statistics, and probability, pp 281–297
37. Manning CD, Raghavan P, Schtze H (2008) Introduction to information retrieval. Cambridge University Press, Cambridge

## Author Biographies

**Jian Wu** received his B.S. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1998 and 2004, respectively. He is currently an associate professor at the College of Computer Science, Zhejiang University, and visiting professor at University of Illinois at Urbana-Champaign. His research interests include service computing and data mining. He is the recipient of the second-grade prize of the National Science Progress Award. He is currently leading some research projects supported by National Natural Science Foundation of China and National High-tech R&D Program of China (863 Program).



**Liang Chen** received his B.S. degree in computer science from Zhejiang University, Hangzhou, China in 2009. He is currently a Ph.D. candidate in the College of Computer Science, Zhejiang University. His publications have appeared in some well-known conference proceedings and international journals. He received the award of "Excellent Intern" from Microsoft Research Asia in 2010. He also served as a reviewer for some international conferences and journals. His research interests include service computing and data mining.

**Zibin Zheng** received his Ph.D. degree from the department of Computer Science and Engineering, The Chinese University of Hong Kong (CUHK) in 2010. He is currently an associate research fellow at the Shenzhen Research Institute, CUHK. He received Outstanding Thesis Award of CUHK in 2012, ACM SIGSOFT Distinguished Paper Award at ICSE2010, Best Student Paper Award at ICWS2010, First Runner-up Award at 2010 IEEE Hong Kong Postgraduate Student Research Paper Competition, and 2010-2011 IBM PhD Fellowship Award. He served as program committee member of IEEE CLOUD2009, SCC2011-2012, and ICSOC 2012. His research interests include cloud computing, service computing, and software engineering.

**Michael R. Lyu** received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, R.O.C., in 1981; the M.S. degree in computer engineering from University of California, Santa Barbara, in 1985; and the Ph.D. degree in computer science from the University of California, Los Angeles, in 1988. He is currently a Professor in the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong, China. He is also Director of the Video over Internet and Wireless (VIEW) Technologies Laboratory. His research interests include software reliability engineering, distributed systems, fault-tolerant computing, mobile networks, Web technologies, multimedia information processing, and E-commerce systems. He has published over 400 refereed journal and conference papers in these areas. He has participated in more than 30 industrial projects and helped to develop many commercial systems and software tools. He was the editor of two book volumes: Software Fault Tolerance (New York: Wiley, 1995) and The Handbook of Software Reliability Engineering (New York: IEEE and New McGraw-Hill, 1996). Dr. Lyu received Best Paper Awards at ISSRE'98 and ISSRE'2003. Dr. Lyu initiated the First International Symposium on Software Reliability Engineering (ISSRE) in 1990. He was the Program Chair for ISSRE'96 and General Chair for ISSRE'2001. He was also PRDC'99 Program Co-Chair, WWW10 Program Co-Chair, SRDS'2005 Program Co-Chair, PRDC'2005 General Co-Chair, ICEBE'2007 Program Co-Chair, SCC'2010 Program Co-Chair, and DSN'2011 General Chair and served in program committees for many other conferences including HASE, ICECCS, ISIT, FTCS, DSN, ICDSN, EUROMICRO, APSEC, PRDC, PSAM, ICCCN, ISESE, and WI. He has been frequently invited as a keynote or tutorial speaker to conferences and workshops in the United States, Europe, and Asia. He has been on the Editorial Board of the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, the IEEE TRANSACTIONS ON RELIABILITY, the Journal of Information Science and Engineering, and Software Testing, Verification & Reliability Journal. Dr. Lyu is an IEEE Fellow and an AAAS Fellow for his contributions to software reliability engineering and software fault tolerance. Dr. Lyu is also a Croucher Senior Research Fellow.

**Zhaohui Wu** received the PhD degree in computer science from Zhejiang University, Hangzhou, China, in 1993. From 1991 to 1993, he was a joint Ph.D. student in the area of knowledge representation and expert system with the German Research Center for Artificial Intelligence, Kaiserslautern, Germany. He is currently a Professor at the College of Computer Science, and the Director of the Institute of Computer System and Architecture, Zhejiang University. His research interests include intelligent transportation systems, distributed artificial intelligence, semantic grid, and ubiquitous embedded systems. He is the author of four books and more than 100 referred papers. Dr. Wu is a Standing Council Member of China Computer Federation (CCF), Beijing. Since June 2005, he has been the Vice Chairman of the CCF Pervasive Computing Committee. He is also on the editorial boards of several journals and has served as a Program Chair for various international conferences.