http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (5): 967-974, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.967.974

A Semantic Approach of Service Clustering and Web Service Discovery

YuYue Du, Yong Jun Zhang and Xing Lin Zhang
College of Information Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China

Abstract: Web service discovery has always been a hot issue in the research field of Web services. In this study, services are grouped into functionally similar service clusters through calculating semantic similarity with WordNet. A Concept Position Vector model of service clusters is proposed, which can sharply cuts in the number of services that do not completely match the service requests, thus can quickly build up the set of candidate services. Therefore, the time efficiency of the service discovery can be improved compared with the general cluster-based method of service discovery.

Key words: Service cluster, semantic similarity, wordNet, service discovery, concept position vector

INTRODUCTION

Nowadays, as SOA (Service-oriented Architecture) has been the main driving force for web application development, the types and number of web services grow rapidly. Thus, finding appropriate services quickly and accurately is considered as hard as searching a needle in the haystack (Garofalakis *et al*., 2004). Many efforts have been made to settle this problem and applying service clustering technique for service discovering is a mainstream idea in recent years. Generally speaking, the process of clustering services starts from parsing web service describing documents (such as WSDL) and extracting features and then groups web services into functionality-based clusters according to a particular methods for clustering (Elgazzar *et al*., 2010).

The main method for service clustering focuses on the similarity among services. Text mining methods are applied to extract features from WSDL in order to group services (Liu and Wong, 2009). And the frequency of keywords that appear in WSDL is used to measure the similarity (Nayak and Lee, 2007). To cluster services, scholars propose a method that combines the techniques of Hyperclique Paterns Disovery and LSI (Latent Semantic Indexing) (Plebani and Pernici, 2009). In order to improve the cohesiveness of the service cluster, someone considers not only the functions of services but also the process inside services. For a given service request, a service matchmaker compares the functionality and process consistency among the candidate services, which can improve the accuracy of matchmaking (Sun and Jiang, 2008). However, this study uses the method that comparing request with services in service clusters one by one, which ignores the similarity of services in a particular service cluster.

Overall, the present cluster-based works largely focus on the methods for clustering web services, but neglect a further study on the description and formalization of service clusters after clustering. In addition, these works do not specify clearly the process for service discovery.

Considering the similarity of services included in a service cluster, a model of Concept Position Vector (CPV) is provided to refine service clusters. It can largely improve the time efficiency of service discovery through clustering services and refine service clusters prior to matching services with a given service request. The overview of our study can be seen through Fig. 1.

This study firstly parsed the WSDL files and calculated the functional similarity of services with WordNet, then clustered services into functionally similar service clusters. And a CPV model was proposed to refine service clusters.

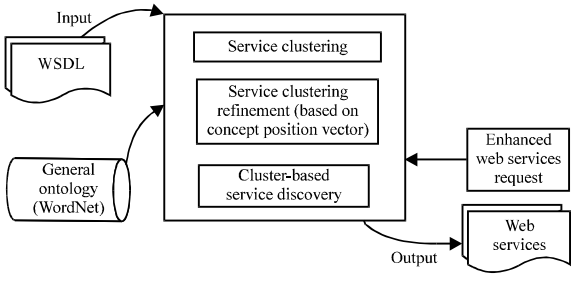

Fig. 1: Schematic block diagram of cluster-based service discovery

Corresponding Author: YongJun Zhang, College of Information Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China

BASIC KNOWLEDGE OF WEB SERVICE

According to WSDL documents, a service contains more than one operation and each operation provides a function, about 75% WSDL documents contain more than two operations (Deng *et al*., 2009). Thus, these operations should be considered when we cluster services.

BASIC DEFINITION

Definition 1: Web service: A web service is a four-tuple:

$$S = (SID, SName, SF, \{Op_m\})$$

where, SID is the unique identifier of the service, SName is the service name which is often a compound word composed of several words with figures or identifiers (such as"-","_") according to service producers' naming conventions.

SF is the functional description in natural language provided by service provider when publishing services. {Opm} is the set of operations in the service. In other words, it is the set of the functions that the service provides. Each operation can be described by S.Op in Defination2.

Definition 2: Service operation: An operation in a web service is a three-tuple:

$$S.Op = (OpID, OpName, \{In_k, Out_l\})$$

where, OpID is the unique identifier of the operation, OpName is the operation name, often a compound word that is similar to service names, $\{In_k, Out_l\}$ present the interface of service functionality that is composed by the input parameters and output parameters of the operation. And each parameter can be described by:

- **S.Op.In_k:** (ParaName, type) defines an input parameter with the name of the parameter and the type
- **S.Op.Out:** (ParaName, type) defines an output parameter in the same way

SEMANTIC SIMILARITY MEASUREMENT

In this study, we use WordNet (Miller, 1995) to calculate the semantic similarity between ontology concepts to group services into functionally similar clusters. Here, we apply the design of public ontologies such as WordNet instead of a domain-ontology for service annotation and similarity evaluation. This is because, on the one hand, different services may come from different domains; however, no public ontology is available now. On the other hand, WordNet, a natural language semantic database combined current psycholinguistic and human lexical items, has enough concepts for annotation. Although some efforts on ontology construction are increasing, such as Bootstrapping Ontologies for Web Services provided by (Segev and Sheng, 2012), we do not apply these methods due to the immaturity.

The factors that influence the semantic similarity between concepts mainly include: the shortest path and the depth. We prefer the methods proposed by (Li *et al*. (2003) that consider both the two factors for semantic similarity computing. The similarity between two concepts C_i and C_j is calculated according to Eq. 1:

$$Sim(C_i, C_j) = e^{-\alpha len(Ci,Cj)} \times \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}} \quad (1)$$

where, len (C_i, C_j) presents the shortest path between the two concepts; h is the depth of the Least Common Ancestor in the ontology hierarchical structure and $\alpha, \beta > = 0$ are the regulating parameters of len(C_i, C_j)and h. Particularly, Li *et al*. (2003) set $\alpha = 0.2$, $\beta = 0.6$ in which case that can bring the perfect effect according to their research.

As shown in Fig. 2, for a fragment of the semantic hierarchy of WordNet, the similarity between two concepts (car,bicycle) (denoted by Sim(car,bicycle)) is considered, and the shortest path between them is car-transportation-bicycle. Thus len(car,bicycle) = 2 and the Least Common Ancestor is *Transportation* whose depth in the actual hierarchical structure of WordNet is 11, and h = 11. Therefore:

$$Sim(car, bicycle) = e^{-0.2*2} \times \frac{e^{0.6*11} - e^{-0.6*11}}{e^{0.6*11} + e^{-0.6*11}} = 0.67$$

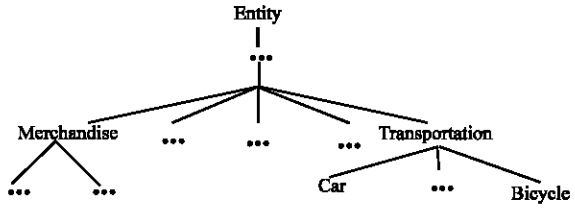

Fig. 2: A fragment of the semantic hierarchy of WordNet

WEB SERVICE CLUSTERING

Process for service clustering

Features mining from web service files: In this study, we use the tool of WSDL4J developed by IBM that provides standard WSDL parsing interfaces to parse WSDL files and mining four types of features, namely, the service name, the operation name, the input parameters and output parameters of the operation. These features represent the functions of the service.

Preprocess for clustering

Tokenization: Due to the naming conventions, the four types of features we extract from WSDL files mostly are composed by several words, for example, ShoppingmallMaxpricedigital-videoService, thus name similarity can be calculated only after a tokenization step which decomposes a given name into its terms. The set of terms will be actually compared to obtain the similarity among names. For this reason, we take the step of tokenization after feature mining and use blank spaces to segregate terms in the same set (Table 1).

Stemming: Stemming technology is often used in Information Retrieval (IR), as many English words are variations of the same lemma. For example, "wheel" is a lemma for "wheeled", "wheeling", "wheeler". We apply the Port Stemmer to trip word endings and return lemmas. For example, "shopping" returns "shop". Through this step, inflections of nouns, conjugations of verbs, and adjectives are recognized, thus can reduce semantic ambiguity.

Stop list: It is also a common technology in the field of IR. Applying stop list, we do stop words removing, thus can filter out terms with less information value. For example, there is a parameter "Lecturer-in-academiaService", and after the step of tokenization, we should add the term "in" to stop list, and returns {lecturer academia service}.

Assignment in bipartite graphs of parameters: Generally speaking, the functional similarity of services relies on both the name of the services and all their operations; whereas the valuation of similarity of operations should take into account both the operation name and all their parameters. As an operation has more than one input parameters and output parameters, so we should pare parameters to realize which is the maximum similarity between the elements included in the two sets we are comparing.

In this study, we want to obtain the global maximum similarity through pairing the elements in the two set, not the local maximum which relies on the order that the comparisons of elements from the two set occur. For example, when applying the method of local maximum, the result of paring illustrated in Fig. 3 is (<i1,i`3>,<i2,i`2>,<i3,i`4>).However, Sim(<i1,i`3>) = 0.8, which is less than Sim(<i2,i`3>) = 1.0.

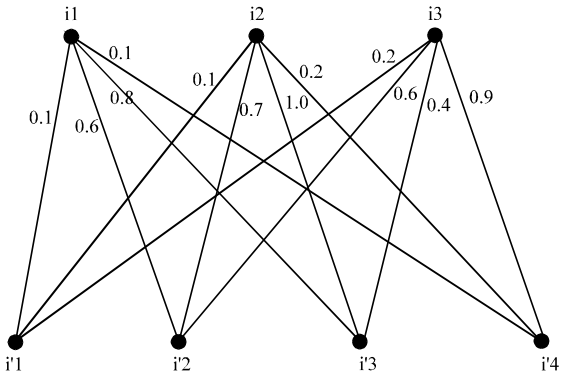

Fig. 3: Graphical representation of bipartite graphs of parameter

Table 1: Rules of tokenization

Rules	Original term	Tokenized version
Rule 1	Personal bicycle	Personal bicycle
Rule 2	Shopping mall	Shopping mall
Rule 3	Calendar-date	Calendar date
Rule 4	4Wheeled car	Wheeled car

Many efforts have been made to solve the problem of parameter pairing by applying the algorithms of assignment in bipartite graphs. In this study, we design a method based on this idea for solving this problem.

Algorithm 1: Assignment in bipartite graphs of parameters

Input: a set of input parameters from one operation in a service $\{S.Op_m.In\}$ and another set of input parameters from one operation in another service $\{S.Op_k.In\}$, a given threshold v.

Output: the set of parameter pairs

1: local a similarity values matrix $a[|S.Op_{m,k}.In|][|S.Op_k.In|]$,
2: For i -->$\{S.Op_m.In\}$, j -->$\{ S.Op_k.In\}$
3: a[i][j] = Sim(<i,j>) // Sim(<i,j>) is calculated according to formula(1);
4: Picking up the maximum of each row in the matrix, $a_{max}[i][j]$;
5: If ($a_{max}[i][j]$> =v) then add $a_{max}[i][j]$ into MaxSim
6: Else delete the row that $a_{max}[i][j]$ lies in
7: /*If there exits more than one elements in the MaxSim that come from the same column of the matrix, then changed the value of the one of lower values into the value of 0, delete it from the MaxSim, then choose the maximum of the row in the matrix again*/
8: For each $a_{max}[i][j]$ in MaxSim
9: If(j_{larger} --> $a_{maxlarger}[i_{larger}][j_{larger}]$)==($j_{smaller}$ --> $a_{maxsmaller}[i_{smaller}][j_{smaller}]$))
10: $a_{maxsmaller}[i_{smaller}][j_{smaller}]$ = 0;
11: delete $a_{maxsmaller}[i_{smaller}][j_{smaller}]$ from MaxSim
12: picking up the maximum of the row $a_{maxsmaller}[i_{smaller}][j_{smaller}]$ lies in, repeat 5),6)
13: Chose the elements that corresponding to rows and columns and pair them, then return the set of parameter pairs.

We can get the number of successfully paired parameters from Algorithm 1 through |MaxSim|(the length of MaxSim).

For example, in the matrix described above, if we set the value of ν is 0.6, the maximums of each row in the matrix are a[0][2] = 0.8, a[1][2] = 1.0, a[2][3] = 0.9 but a[0][2], a[1][2] are from the same column(j = 2), and a[0][2]< a[1][2], so we set a[0][2] = 0, and choose the maximum from the first row in the matrix again, that is a[0][1] = 0.6, equals to ν, so MaxSim = [0.6,1.0,0.9]. Then we get the set of parameter pairs {<i 1, i`2>, i2,i`2 >,<i3,i`4>}.

Functional similarity of web services

Similarity of the set of input/output parameters: Given two operations (x and y) from different services, based on the parameter pairing, we can get the similarity of the set of inputs (or outputs), which is showed by Formula (2):

$$Sim(<In_x, In_y>) = num/(|In_x| + |In_y| - num) \quad (2)$$

where, num is the number of successfully paired parameters. In the previous example:

$$Sim(<In_x, In_y>) = 3/(3+4-3) = 0.75.$$

And we deal with the outputs in the same way as with inputs.

Operation similarity function: Operational similarity relies on three functions: an operational name similarity Sim (<Name $_{si.opk}$, Name $_{sj.opl}$ >), the similarity of input parameters Sim (<In $_{si.opk}$, In$_{sj.opl}$ >) and the similarity of output parameters Sim (<Out$_{si.opk}$, Out $_{sj.opl}$>, as Formula (3) shows:

$$OpSim(<S_i.Op_k, S_j.Op_l>) = \omega 1 * Sim(<Name_{si.opk}, Name_{sj.opl}>) + \omega_2 * Sim(<In_{si.opk}, In_{sj.opl}>) + \omega_3 * Sim(<Out_{si.opk}, Out_{sj.opl}>) \quad (3)$$

where, ω_1, ω_2, $\omega_3 \in [0, 1]$, as weights for operation name, input parameters and output parameters, in addition, $\omega_1+\omega_2+\omega_3 = 1$. Generally, the selection of the weight coefficients, to some extent, is a key challenge for relevant research. It is somewhat subjective at present.

Web service similarity function ssim(s_i,s_j): As one service often contains more than one operation, so we consider the average of all operational similarity, which is Formula (4):

$$AvgOpSim(S_i,S_j) = \Sigma\, sim <S_i.Op_k, S_j.Op>/n \quad (4)$$

where, n = min {$|S_i.Op_k|$, $|S_j.Op_l|$}.

The functional similarity of services relies on two main functions: a service name similarity function Sim (<Name$_{si}$, Name $_{sj}$>) and the average operational similarity function *AvgOpSim(<$S_i.Op_k$, $S_j.Op_l$ >)*. The weight for service name is the same as operation name, that is ω_1. Web service similarity functional similarity, defined by Ssim(s_i,s_j), is evaluated by Formula (5):

$$Ssim(si,sj) = \omega 1 * Sim(<Name_{si}, Name_{sj}>) + (1- \omega 1) * AvgOpSim(S_i, S_j) \quad (5)$$

Algorithm for service clustering: A bottom-up algorithm of hierarchical clustering is applied to service clustering based on semantic similarity.

We firstly create a service cluster and randomly add one service Su into it with viewing Su as the center of this cluster. Then we scroll the rest set of services, and select the ones that satisfy a certain similarity condition through calculating the similarity with Su and add them into the same cluster which Su belongs to. If the similarity cannot satisfy the conditions, then we build up a new cluster and add this service in. The above steps repeat until each of the services can be mapped into a service cluster. The algorithm for service clustering is described in detail as Algorithm 2 shows.

Algorithm 2: Service clustering

Input: All services in service register, the threshold for service functional similarity (ν).
Output: The set of service clusters Sc_{set} = {SC_1, SC_2, ..., SC_k}

1: Local SC_{set}=ø;
2: For each service S_u that does not belong to any service cluster, build a new cluster{S}
3: Sc_{set} = SC_{set}∪{SC_u}; //add this cluster to SC_{set}
4: For each service S_j that do not belong to any service clusters
5: SSim (S_u,S_j) // compare S_u with all the other services that also do not belong to any clusters
6: If (SSim (S_u,S_j) >=ν) then add S_j to {SC_u}
7: Repeat the steps of 2) and 3);
8: Return Sc_{set};

SERVICE DISCOVERY BASED ON SERVICE CLUSTERS

After clustering, the services in the service registry can be mapped into different clusters that are loosely-coupled and highly-cohesive. In order to improve the time efficiency in service discovery, we take advantage of the functional similarity of services in the same service cluster. Therefore we take a further step to refine service clusters. Then based on clusters, we improve the time efficiency of service discovery. The whole process of our study is depicted by Fig. 4.

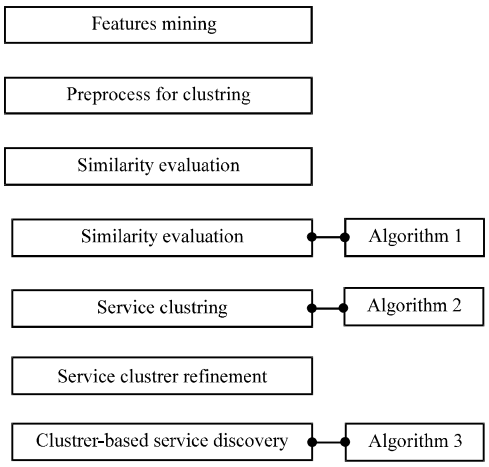

Fig. 4: the whole process of service discovery

Service cluster refinement

Concept annotation: To eliminate the semantic ambiguity of terms included in the service cluster, this study apply Concept annotation technology Plebani and Pernici (2009) that is annotating each term with a concept from a public ontology (in this study ,we refer to WordNet as it is domain-independent and it has enough concepts to annotate terms parsed from various services).

After the step of concept annotation, the expended descriptions of services and operations can be got:

$$S^c = (SID, SName, Concept, SF, \{Op_m^c\})$$

$$S^c.Op^c = (OpID, OpName, Concept, \{In_k^c, Out_l^c\})$$

$$S^c.Op^c.In_k^c = (ParaName, Concept, type)$$

$$S^c.Op^c.Out_l^c = (ParaName, Concept, type)$$

Refinement of service clusters

Definition 3: Service cluster: A web service is a five-tuple SC = (SCID, SCF, {SCInP^c}, where, {SCOutP^c}, {S^c.Op^c}):

SCID	:	The unique identifier of the service cluster
SCF	:	The functional description in natural language that is added by hand according to the functional description of the services included in the cluster
{SCInP^c}	:	The concept set of input after concept annotation for each input
{SCOutP^c}	:	The concept set of output after concept annotation for each output
{S^c.Op^c}	:	The set of all operations which also takes the step of concept annotation included in the service cluster

Definition 4: Concept position vector, CPV: There is a set of concepts $\{c_1, c_2, ..., c_n\}$, if $cv_1, cv_2, ..., cv_n$ is a rank of position number of elements in this concept set, then we call $cV = (cv_1,cv_2,...,cv_n)$ as Concept Position Vector of this concept set.

We can get Input/Output CPV of service cluster that correspond to the concept sets of inputs/outputs:

$$cV(SCInP^c) = (cv_1,cv_2,...,cv_n),\ n>0 \text{ and } n = |SCInP^c|$$

$$cV(SCOutP^c) = (cv_1,cv_2,...,cv_m),\ m>0 \text{ and } m = |SCOutP^c|$$

As Input/Output concept set of service cluster is the union set from Input/Output concept set of all operations in the service cluster, thus:

$$\forall Sc^c.Op^c.In_k^c, \exists SCInP_{k^*}{}^c \in \{SCInP^c\}$$
$$\text{and } <Sc.Op^c.In_k^c,\ SCInP_k{}^{*c}>$$

$$\forall S^c.Op^c.Out_l^c,\ \exists SCOutP_{l^*}{}^c \in \{SCOutP^c\}$$
$$\text{and } <S^c.Op^c.Out_l^c, S^c.Op^c.Out_{l^*}{}^c>$$

With Input/Output CPV of service cluster, we can get Input/Output CPV of each operation included in the service cluster:

$$cV(S^c.Op^c.In_k^c) = k^*,\ \exists\ SCInP_{k^*}{}^c \in \{SCInP^c\}$$
$$\text{and } < S^c.\ Op^c.\ In_k^c,\ SCInP_{k^*}{}^c> \| cV(SCInP_{k^*}{}^c) = k^*$$

$$cV(S^c.Op^c.Out_l^c) = l^*,\ \exists\ SCOutP_{l^*}{}^c \in \{SCOutP^c\}$$
$$\text{and } < S^c.\ Op^c.Out_l^c, S^c.Op^c.Out_{l^*}{}^c> \| cV(SCInP_{l^*}{}^c) = l^*$$

Then we put the results of Input/Output CPV for each operation into the registry.

For example, there is an input concept set of a service cluster, expressed as (c_1, c_2, c_3), and $cV(c_1) = 1, cV(c_2) = 2, cV(c_3) = 3$; if there is an service operation ($S^c.Op_1{}^c$) included in this service cluster that contains two concepts, c_3 and c_1. Then the Input CPV of $S^c.Op^c{}_1$ is $cV(S^c.Op_1{}^c.In^c) = (1,3)$.

Service discovery process

Service request: The goal of service discovery is finding the suitable services (or more precisely speaking, finding the suitable service operations) that meet users' requirements.

The formal description of service request is given by Definition 4.

Definition 5: Service request: A web service request is a three-tuple:

$$R = (\{RIn\}, \{ROut\}, [r'])$$

where, {Rin} is the set of input parameters of service request, {ROut} is the set of output parameters of service request and r' , an optional parameter, is the matching threshold of services comparing with the request, which is provided by user ,or a default value provided by the system.

In order to get optimal performance of cluster-based service discovery, concept annotation methods should be used to not only services of intra-cluster but the service request. Thus, we can get the formal semantic description for the enhanced request: $R^c = (\{RIn^c\}, \{ROut^c\}, [r'])$, where $\{RIn^c\}$is the concept set of inputs; $\{ROut^c\}$ is the outputs concept set.

Algorithm for service discovery: Once the service request is defined, the common method usually takes so much time in matching it with all of the services in the registry that it is often inefficiency.

In this study, we do the service clustering and the refinement of service clusters off-line which does not increase the time spent on searching in runtime as Algorithm 3 shows.

Algorithm 3: Service discovery

Input: a service request R^c and the set of service clusters SC_{Set};
Output: the set of candidate service-operations that potentially satisfy service request CandidateOp;

1: Local CandidateInOp = ∅, CandidateOutOp = ∅, and CandidateOp=∅;
2: Randomly choose a service cluster SC_i from SC_{Set}
3: /* whether the outputs concept of the request $\{ROut^c\}$ all can be provided by the outputs concept set of the cluster $\{SC_i OutP^c\}$*/
4: If ($\exists o \in \{ROut^c\}$, and $o \notin \{SC_i OutP^c\}$) then go to 2);
5 else { get CPV of request outputs cV($ROut^c$);
6: for each service S in SC_i
7: if (cV($S^c.Op_k^c.Out^c$) = = cV($ROut^c$))
8: CandidateOutOp = CandidateOutOp∪ $S^c.Op_k^c$;
9: If (CandidateOutOp==∅) then go to 2);
10: }
11: /* whether the inputs concept of the request$\{RIn^c\}$ all can be provided by the inputs concept set of service $\{SC_i InP^c\}$*/
12: Get CPV of request inputs *cV(RIn^c)*;
13: for each service *S* in SC_i
14: if cV($S^c.Op_k^c.In^c$) = cV(RIn^c)
15: CandidateInOp = CandidateInOp∪ $S^c.Op_k^c$;
16: If (CandidateInOp==∅) then go to 2);
17: CandidateOp = CandidateOutOp∩ CandidateInOp;
18: Return CandidateOp.

This algorithm begins with randomly choosing a service cluster SC_i from service clusters set SC_{Set} , then from 4 to 10, it matches outputs of this cluster with the given request and from 12 to 16, it matches the inputs with the request. Step 5 and 12 are proposed to get CPV of request outputs/inputs (cV($ROut^c$, cV(RIn^c)). From 6 to 8, it compares each service-operations in the cluster with cV($ROut^c$),if operations meet the conditions that the CPV of the operation outputs equals to that of request (in step 7), then it will add the operation to CandidateOutOp which means the candidate service-operations that can supply the outputs of the request (showed in step 8). From 13 to 16, it compares each service-operations in the cluster with cV (RIn^c). If this cluster cannot meet the conditions, it will choose another cluster (as depicted in step 2). At last, it will return the set of candidate service-operations in this cluster that potentially match the request.

EXPERIMENT AND EVALUATION

Data preparation: OWL-TC (http://projects.semwebcentral .org/projects/owls-tc) is the OWL-S service retrieval test collection, and the newest version is OWL-TC4 which we adopt in our context. It provides 1083 Web services specified with OWL-S covering 9 application domains, such as food, economy, communication, travel, medical care, weaponry, education, geography and simulation, with 42 test queries. Since our method is based on parsing WSDL, the tool OWLS2WSDL is used to derive WSDL from OWL-S documents. After parsing all these WSDL files, we get 1075 services, 1075 operations, 1533 input parameters, 1606 output parameters.

The experiments run on a Windows XP PC with 1.60 GHz processor, including Exlips6.1, WordNet2.1, and My SQL 5.0.

RESULTS

Threshold for similarity: This phase aims at identifying for which value of threshold for similarity (defined by ν) to cluster services that can obtain the best performances. For the sake of simplicity, we randomly select 115 services to do these experiments in this phase.

We set the similarity threshold to four kinds of cases: ν = 0.5, 0.6, 0.7, 0.8. In the case of ν = 0.5, we divide the 115 services into 5 service clusters with the number of services in each cluster is respectively 7, 7, 21, 66, 13. In the case of ν = 0.6, we divide the 115 services into 10 service clusters with the number of services in each cluster is respectively, 7, 3, 12, 13, 12, 10, 38, 11, 4, 4. In the case of ν = 0.7, there are 10 clusters with the respective number of services containing 2,5,3,10,7,8,39,33,2,6. In the case of ν = 0.8, there are 11 clusters with the respective number of services 1, 6, 2, 2, 3, 10, 8, 3, 35, 23, 22.

We test the time spending on service discovery for the same request in the four cases, and we do this for 20 times with recording results every time. After figuring up

the average response time of service discovery, we find that in the case of $\nu = 0.7$ that can get better performance than other cases, as Table 2 shows.

Evaluating the efficiency of service discovery: General methods for service discovery based on clustering often compare the request with all of the services in the registry without taking the functional similarity of service cluster into account, such as Sun and Jiang (2008) did. In contrast, our method for service discovery based on refining services clusters with Concept Position Vector after clustering.

In this experiment, we carry out 100 times test of service discovery with OWLS-TC. We specify the threhold value for similarity as $\nu = 0.7$ and the matching threshold in the request as $r' = 0.7$. Figure 3 shows the results of our methods (defined by CPVCluster-SD) compared with the work of Sun and Jiang (2008) (defined by GeneralCluster-SD).

From Fig. 5, we can see that our method can reduce 30-60% respond time compared with the general method for service discovery based on clustering. And, with the number of service increasing, the respond time using the General Cluster-SD increases dramatically compared with using our method and the better stability of our method is illustrated. In other words, our method can effectively improve time efficiency of service discovery. In addition, considering the precision/recall, our method has the same performance with GeneralCluster-SD.

Table 2: Average response time in different cases of similarity threshold

Threshold for similarity	Time of service discovery (msec)
$\nu = 0.5$	290
$\nu = 0.6$	264
$\nu = 0.7$	251
$\nu = 0.8$	261

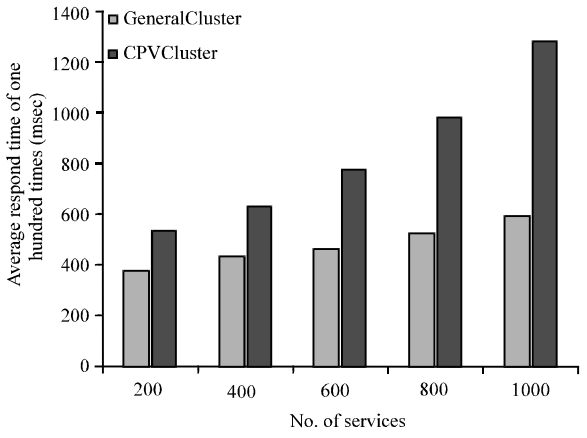

Fig. 5: The comparison of respond time of service discovery (SD)

CONCLUSIONS

In this study, we present an approach for service discovery based on service clustering and refining service clusters. All in all, we use WordNet to compute semantic similarity, and cluster services based on functional similarity, then refine service clusters with Concept Position Vector.

In the future, we will focus on service composition based on service clusters. However, many problems are needed to be resolved, such as, service quality problem which we do not considered in this study, the step of to kenization for combination-terms will need to optimize, and how to maintain the service clusters should also be taken into consideration in the future.

ACKNOWLEDGMENTS

This work is supported by the National Basic Research Program of China under grant 2010CB328101; the National Natural Science Foundation of China under grants 61170078 and 61173042; the Doctoral Program of Higher Education of the Specialized Research Fund of China under grant 20113718110004; the Scientific and Technological Developing Program of Shandong Province of China under grants 2011GGX10114 and 2012G0020120; the SDUST Research Fund of China under Grant 2011KYTD102 and the project of Shandong Province Higher Educational Science and Technology program under Grant J12LN11.

REFERENCES

Deng, S., Z. Wu, J. Wu, Y. Li and J. Yin, 2009. An efficient service discovery method and its application. Int. J. Web Serv. Res., 6: 94-117.

Elgazzar, K., A.E. Hassan and P. Martin, 2010. Clustering WSDL documents to bootstrap the discovery of web services. Proceedings of the 2010 IEEE International Conference on Web Services, July 05-10, 2010, Miami, FL., pp: 147-154.

Garofalakis, J., Y. Panagis, E. Sakkopoulos, and A. Tsakalidis, 2004. Web service discovery mechanisms: Looking for a needle in a haystack? Proceedings of the International Workshop on Web Engineering, Hypermedia Development and Web Engineering Principles and Techniques: Put Them in Use, in Conjunction with ACM Hypertext, August 9-13, 2004, Santa Cruz.

Li, Y., Z.A. Bandar and D. McLean, 2003. An approach for measuring semantic similarity between words using multiple information sources. IEEE Trans. Knowledge Data Eng., 15: 871-882.

Liu, W. and W. Wong, 2009. Web service clustering using text mining techniques. Int. J. Agent-Orient. Software Eng., 3: 6-26.

Miller, G.A., 1995. WordNet: A lexical database for English. Commun. ACM, 38: 39-41.

Nayak, R. and B. Lee, 2007. Web service discovery with additional semantics and clustering. Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, November 2-5, 2007, Fremont, CA., pp: 555-558.

Plebani, P. and B. Pernici, 2009. URBE: Web service retrieval based on similarity evaluation. IEEE Trans. Knowl. Data Eng., 21: 1629-1642.

Segev, A. and Q.Z. Sheng, 2012. Bootstrapping ontologies for web services. IEEE Trans. Serv. Comput., 5: 33-44.

Sun, P. and C.J. Jiang, 2008. Using service clustering to facilitate process-oriented semantic web service discovery. Chinese J. Comput., 31: 1340-1353.