WSExpress: A QoS-Aware Search Engine for Web Services

Yilei Zhang, Zibin Zheng, and Michael R. Lyu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
{ylzhang,zbzheng,lyu}@cse.cuhk.edu.hk

Abstract

Web services are becoming prevalent nowadays. Finding desired Web services is becoming an emergent and challenging research problem. In this paper, we present WSExpress (Web Service Express), a novel Web service search engine to expressively find expected Web services. WSExpress ranks the publicly available Web services not only by functional similarities to users' queries, but also by non-functional QoS characteristics of Web services. WSExpress provides three searching styles, which can adapt to the scenario of finding an appropriate Web service and the scenario of automatically replacing a failed Web service with a suitable one. WSExpress is implemented by Java language and large-scale experiments employing real-world Web services are conducted. Totally 3,738 Web services (15,811 operations) from 69 countries are involved in our experiments. The experimental results show that our search engine can find Web services with the desired functional and non-functional requirements. Extensive experimental studies are also conducted on a well known benchmark dataset consisting of 1,000 Web service operations to show the recall and precision performance of our search engine.

1. Introduction

With a set of standard protocols, i.e., SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), and UDDI (Universal Description, Discovery and integration), Web services provided by different organizations can be discovered and integrated to develop applications [2]. With the growing number of Web services in the Internet, many alternative Web services can provide similar functionalities to fulfill users' requests. Syntactic or semantic matching approaches based on services' tags in UDDI repository are usually employed to discover suitable Web services [13]. However, discovering web services from UDDI repositories suffers several limitations. First, since UDDI repository is no longer a popular style for publishing Web services, most of the UDDI repositories are seldom updated. This means that a significant part of information in these repositories is out of date. Second, arbitrary tagging methods used in different UDDI repositories add to the complexity of searching Web services of interest.

To address these problems, an automated mechanism is required to explore existing Web services. Considering that WSDL files are used for describing Web services and can be obtained in several ways other than UDDI repositories, several WSDL based Web service searching approaches are proposed. Such as *Binding Point*[1], *Grand Central*[2], *Salcentral*[3], and *Web Service List*[4]. However, these engines only simply exploit keyword-based search techniques which are obviously insufficient for catching the Web services' functionalities. First, keywords cannot represent Web services' underlying semantics. Second, since a Web service is supposed to be used as part of the user's application, keywords cannot precisely specify the information user needs and the interface acceptable to the user. In this paper, we employ not only keywords but also operation parameters to comprehensively capture Web service's functionality.

In addition, Web services sharing similar functionalities may possess very different non-functionalities (e.g., response time, throughput, availability, usability, performance, integrity, etc.). In order to effectively provide personalized Web service ranking, it is requisite to consider both functional and non-functional characteristics of Web services. Unfortunately, the Web service search engines mentioned above cannot distinguish the non-functional differences between Web services.

QoS-driven Web service selection is a popular research problem [1, 9, 15]. A basic assumption in the field of selection is that all the Web services in the candidate set share identical functionality. Under this assumption, most of the selection approaches can only differentiate among Web

[1] http://www.bindingpoint.com/
[2] http://www.grandcentral.com/directory/
[3] http://www.salcentral.com/
[4] http://www.webservicelist.com/

978-0-7695-4128-0/10 $26.00 © 2010 IEEE
DOI 10.1109/ICWS.2010.20

services's non-functional QoS characteristics, regardless of their functionalities. While these QoS-driven selection approaches are directly employed to Web service search engines, several problems will arise. One is that Web services whose functionalities are not exactly equivalent to the user searching query are completely excluded from the result list. Another problem is that Web services in the result list are ordered only according to their QoS metrics, while combining both functional and non-functional attributes is a more reasonable method.

To address the above issues, we propose a new Web service discovering approach by paying respect to functional attributes as well as non-functional features of Web services. A search engine prototype, WSExpress, is built as an implementation of our approach. Experimental results show that our search engine can successfully discover user-interested Web services within top results. In particular, the contributions of this paper are three-fold:

- Different from all previous work, we propose a brand new Web service searching approach considering both functional and non-functional qualities of the service candidates.
- We conduct a large-scale distributed experimental evaluation on real-world Web services. 3,738 Web services (15,811 operations) located in 69 countries are evaluated both on their functional and non-functional aspects. The evaluation results show that we can recommend high quality Web services to the user. The precision and recall performance of our functional search is substantially better than the approach in previous work [11].
- We publicly release our large-scale real-world Web service WSDL files and associated QoS datasets[5] for future research. To the best of our knowledge, our dataset is the first publicly-available real-world dataset for functional and non-functional Web service searching research.

The rest of this paper is organized as follows: Section 2 introduces Web service searching scenarios and the system architecture. Section 3 presents our QoS-aware searching approach. Section 4 describes our experimental results. Section 5 introduces related work and Section 6 concludes the paper.

2. Preliminaries

2.1. A Motivating Example

Figure 1 shows a common Web service query scenario. A user wants to find an appropriate Web service which contains operations that can be integrated as part of the user's application. The user needs to specify the functionality of a suitable operation by filling the fields of keywords, input and output. Also the user may have some special requirements on service quality, such as the maximum price. These personal requirements can be represented by setting the QoS constraint field. The criticality of different quality criteria for a user can be defined by setting the QoS weight field.

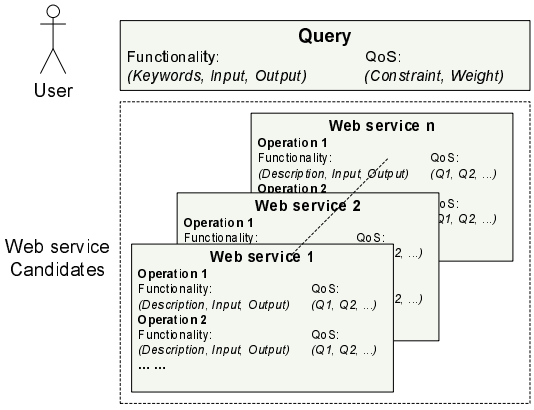

Figure 1. Web Service Query Scenario

A lot of Web services can be accessed over the Internet. Each service candidate provides one or more operations. Generally, these operations can be described in the structure shown in Figure 1. Each operation includes a name, the parameters of input and output elements, and the descriptions about the functionality of this operation as well as the Web services it belongs to in its associated WSDL document. The service quality associated with this operation is represented by several criteria values, e.g., Q1, Q2 in Figure 1.

Table 1 shows Web service query examples. In query 1, a user wants to find a Web service that can provide appropriate operations for displaying prices of different types and brands of cars. The input information provided by the user for that particular operation is the types and names of cars. This query is structured into three parts: *keywords*, *input* and *output*. The *keywords* part defines in which domain is the query about. In this example, the user concerns about the domain "car". The *input* part contains "name" and "type" since they can be provided by the user. The *output* part is set as "price" to specify the information the user wants to obtain from an appropriate operation.

In Table 2 we enumerate three possible results for the user's search query. Web service 1 provides one operation *CarPrice* and this operation's functionality is almost the same as what the user specifies in the query. In addition, the service quality meets the user's requirements. Web service 2 provides operation *AutomobileInformation*. Operation *AutomobileInformation* can provide many information

[5] http://wiki.cse.cuhk.edu.hk/user/ylzhang/doku.php?id=icwsdata

Table 1. User Query Examples

User Query	Functionality			QoS	
	Keywords	Input	Output	Constraint (C1, C2, C3)	Weight (W1, W2, W3)
query 1	car	name, type	price	(0.5, 0.5, 0.2)	(0.4, 0.4, 0.2)
query 2	weather	city, country	weather	(0.6, 0.3, 0.3)	(0.3, 0.4, 0.3)

Table 2. Web Service Examples

Web Service Name	Operation Name	Input	Output	QoS (Q1, Q2, Q3)
WS 1	CarPrice	name, type	price	(0.8, 0.6, 0.6)
WS 2	AutomobileInformation	name, model	price, color, company	(0.2, 0.4, 0.6)
WS 3	VehicleRecommend	name, model, usage	rent, primecost, provider	(0.6, 0.8, 0.5)

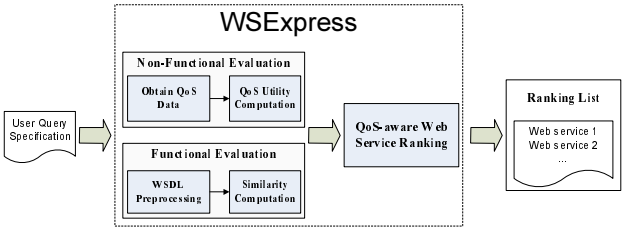

Figure 2. System Architecture

details including the price of the automobiles after invoked with "name" and "model" as input. However, some service quality criteria, such as the service price (Q1) and the response time (Q2), are beyond the user's tolerance. Operation *VehicleRecommend* provided by Web service 3 recommends suitable vehicles for the user to rent. Although its target is to suggest the most suitable vehicle and vehicle rental companies to the user, it can also be invoked for obtaining the prices of cars due to the prime cost information provided. Besides, operation *VehicleRecommend*'s service quality fits the user's constraints and preferences quit well. Among these three Web services, the most suitable Web service is the first one, and another acceptable one is Web service 3, but Web service 2 is not highly suggested due to its service quality. Thus, a reasonable order of the recommendation list for the user's query is Web service 1, Web service 3, and Web service 2.

2.2. System Architecture

Now we describe the system architecture of our Qos-aware Web service search engine. As shown in Figure 2, after accepting a user's query specification, our search engine should be able to provide a practical Web service recommendation list. The search engine consists of three components: non-functional evaluation, functional evaluation, and QoS-aware Web service ranking.

There are two phases in the non-functional evaluation component. In phase 1, the search engine obtains QoS criteria values of all the available Web services. In phase 2, the search engine computes the QoS utilities of different Web services according to the constraints and preferences specified in the QoS part of the user's query.

The functional evaluation component contains two phases. In phase 1, the search engine carries out a preprocessing work on the WSDL files associated to the Web services. This work aims at removing noise and improving accuracy of functional evaluation. In phase 2, the search engine evaluates the Web service candidates' functional features. These features are described by similarities between the functionality specified in the query and the functionality of operations provided by those Web services.

Finally, the search engine combines both functional and non-functional features of Web services in the QoS-aware Web service ranking component. A practical and reasonable Web service recommendation list is then provided as a result to the user's search query.

3. QoS-Aware Web Service Searching

3.1. QoS Model

In our QoS model we describe the quantitative non-functional properties of Web services as quality criteria. These criteria include generic criteria and business specific criteria. Generic criteria are applicable to all Web services like response time, throughput, availability and price, while business criteria such as penalty-rate are specified to certain kinds of Web services.

By assuming m criteria are employed for representing a Web service quality, we can describe the service quality using a QoS vector $(q_{i,1}, q_{i,2}, \ldots, q_{i,m})$, where $q_{i,j}$ represents the j^{th} criterion value of Web service i.

Some QoS criteria values of Web services, such as penalty rate and price, can be obtained from the service providers directly. However, other QoS attributes' values like response time, availability and reliability need to be generated from all the users' invocation records due to the differences between network environments. In this paper,

we use the approach proposed in [16] to collect QoS performance on real-world Web services.

By putting all the Web services' QoS vectors together, we can obtain the following matrix Q. Each row in Q represents a Web service, while each column represents a QoS criterion value.

$$Q = \begin{pmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,t} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,t} \\ \vdots & \vdots & \vdots & \vdots \\ q_{s,1} & q_{s,2} & \cdots & q_{s,t} \end{pmatrix} \quad (1)$$

A utility function is used to evaluate the multi-dimensional quality of a Web service. The utility function maps a QoS vector into a real value for evaluating the Web service candidates. To represent user priorities and preferences, two steps are involved into the utility computation. First, the QoS criteria values are normalized to enable a uniform measurement of the multi-dimensional quality of service independent of their units and ranges. Second, the weighted evaluation on criteria needs to be carried out for representing user's constraints, preference and special requirements.

Normalization In this step each criterion value is transformed to a real value between 0 and 1 by comparing it with the maximum and minimum values of that particular criterion among all available Web service candidates. The maximum value $Q_{max}(k)$ and minimum value $Q_{min}(k)$ of k^{th} criterion are computed as follows:

$$Q_{max}(k) = \max_{\forall j \in [1,n]} q_{j,k} \quad (2)$$

$$Q_{min}(k) = \min_{\forall j \in [1,n]} q_{j,k} \quad (3)$$

The normalized value of $q_{i,j}$ can be represented by $q'_{i,j}$ as follows:

$$q'_{i,j} = \frac{q_{i,j} - Q_{min}(k)}{Q_{max}(k) - Q_{min}(k)} \quad (4)$$

Thus, the QoS matrix Q is transformed into a normalized matrix Q' as follows:

$$Q' = \begin{pmatrix} q'_{1,1} & q'_{1,2} & \cdots & q'_{1,t} \\ q'_{2,1} & q'_{2,2} & \cdots & q'_{2,t} \\ \vdots & \vdots & \vdots & \vdots \\ q'_{s,1} & q'_{s,2} & \cdots & q'_{s,t} \end{pmatrix} \quad (5)$$

Utility Computation Some Web services need to be excluded from the candidate set due to their inconsistency with the user's QoS constraints. Assume a user's constraint vector is $C = (c_1, c_2, \ldots, c_m)$, in which c_i sets the minimum normalized i^{th} criterion value. We will only consider those Web services whose criteria values are all larger than the constraints. In other words, we delete the rows which fail to satisfy the constraints from Q' and produce a new matrix Q^*. For the sake of simplicity, we only consider positive criteria whose values need to be maximized (negative criteria can be easily transformed into positive attributes by multiplying -1 to their values).

A weight vector $W = (w_1, w_2, \ldots, w_m)$ is used to represent user's priorities on preferences given to different criteria with $w_k \in \mathbb{R}_0^+$ and $\sum_{k=1}^{m} w_k = 1$. The final QoS utilities vector $U = (u_1, u_2, \ldots)$ of Web service candidates are therefore can be computed as follows:

$$U = Q^* * W^T \quad (6)$$

in which u_i is the i^{th} Web service QoS utility value within range $[0, 1]$.

3.2. Similarity Search

Now we describe a similarity model for computing similarities between a user query and Web service operations. In this model, a vector $(Keywords, Input, Output)$ is used to represent the functionality part of a user query as well as the functionality part of Web service operations. Particularly, the keywords of a Web service operation are abstracted from the descriptions in its associated WSDL file. Two phases are involved in the similarity search: WSDL preprocessing and similarity computation.

WSDL Preprocessing In order to improve the accuracy of similarity computation for operations and user query in our approach, we first need to preprocess the WSDL files. There are two steps as follows:

1. Identify useful terms in WSDL files. Since the descriptions, operation names and input/output parameters' names are made manually by the service provider, there are a lot of misspelled and abbreviated words in real-world WSDL files. This step replace such kind of words with normalized forms.

2. Perform word stemming and remove stopwords. A stem is the basic part of the word that never changes even when morphologically infected. This process can eliminate the difference between inflectional morphemes. Stopwords are those with little substantive meaning.

Similarity Computation Now we describe how to measure the similarities of Web service operations to a user's query. The functionality part a user's query R_f consists of three elements $R_f = (r^k, r^{in}, r^{out})$. The *keywords* element is a vector $r^k = (r_1^k, r_2^k, \ldots, r_l^k)$, where r_i^k is the i^{th} keyword. Moreover, the *input* element $r^{in} = (r_1^{in}, r_2^{in}, \ldots, r_m^{in})$ and the *output* element $r^{out} = (r_1^{out}, r_2^{out}, \ldots, r_n^{out})$, where r_i^{in} and r_i^{out} are the i^{th} terms of input element and output element respectively. A Web

service operation also consists of three elements $OP_f = (K, In, Out)$. The *keywords* element of operation i is a vector of words $K^i = (k_1^i, k_2^i, \ldots, k_{l'}^i)$. The *input* and the *output* elements are vectors $In^i = (in_1^i, in_2^i, \ldots, in_{m'}^i)$ and $Out^i = (out_1^i, out_2^i, \ldots, out_{n'}^i)$ respectively. Thus, users' queries and Web service operations are described as sets of terms. By applying the TF/IDF (Term Frequency/Inverse Document Frequency) measure [12] into these sets, we can measure the cosine similarity s_i between Web service operation i and a user's query.

3.3. QoS-Aware Web Service Searching

With an increasing number of Web services being made available in the Internet, users are able to choose functionally appropriate Web services with high non-functional qualities in a much larger set of candidates than ever before. It is highly necessary to recommend to the user a list of service candidates which fulfill both the user's functional and non-functional requirements.

To attack the above problem, we propose a novel search engine which can provide the user with brand new searching styles. We define a user search query in the form of a vector $R = (R_f, R_q)$, which contains functionality part R_f and non-functionality part R_q for representing the user's ideal Web service candidate. $R_q = (C, W)$ defines the user's nonfunctional requirements, where C and W set the user's constraints and preferences on QoS criteria separately as mentioned in Section 3.1. Our new searching procedure consists of three styles in the following discussion.

Keywords Specified In this searching style, the user only needs to simply enter the keywords vector r^k and QoS requirements R_q. The keywords should capture the main functionality the user requires in the search goal. In Table 1 as an example, since the user needs price information of cars, it is reasonable to specify "car" or "car, price" as the keywords vector.

Interface Specified In order to improve the searching efficiency, we design the "interface specified" searching style. In this style, the user specifies the expected functionality by setting the input vector r^{in} and/or output vector r^{out} as well as QoS requirements R_q. The input vector r^{in} represents the largest amount of information the user can provide to the expected Web service operation, while the output vector represents the least amount of information that should be returned after invoking the Web service operation.

Similar Operations For a more accurate and advanced Web service searching, we design the "similar operation" searching style by combining above two styles. This style is especially suitable in the following two situations. In the first situation, the user has already received a Web service recommendation list by performing one of the above searching styles. The user decides the Web service to explore in detail, checks the inputs and outputs of its operations, and even tries some of the operations. After carefully inspecting a Web service the user may find that this Web service is not suitable for the applications. However, the user does not want to repeat the time-consuming inspecting process for other service candidates. This style enables the user to find similar Web service operations by only modifying a small part of the previous query to exclude these inappropriate features. In the second situation, the user already integrates a Web service into the application for a particular functionality. However, due to some reason this web service becomes unaccessible. Without requesting an extra query process, the search engine can automatically find other substitutions.

After receiving the users' query, the functional component of WSExpress computes the similarity s_i in Section 3.2 between search query R_f and operations of Web service i, while the non-functional component of WSExpress employs R_q to compute the QoS utility u_i in Section 3.1 of each Web service i.

A final rating score r_i is defined to evaluate the conformity of each Web service i to achieve the search goal.

$$r_i = \lambda \cdot \frac{1}{\log(p_{s_i} + 1)} + (1 - \lambda) \cdot \frac{1}{\log(p_{u_i} + 1)}, \quad (7)$$

where p_{s_i} is the functional rank position and p_{u_i} is the non-functional rank position of Web service i among all the service candidates. Since the absolute values of similarity and service quality indicate different features of Web service and include different units and range, rank positions rather than absolute values is a better choice to indicate the appropriateness of all candidates. $\frac{1}{\log(p+1)}$ calculates the appropriateness value of a candidate in position p for a query. $\lambda \in [0, 1]$ defines how much the functionality factor is more important than the non-functionality factor in the final recommendation.

λ can be a constant to allocate a fixed percentage of the two parts' contributions to the final rating score r_i. However, it is more realistic if λ is expressed as a function of p_{s_i}:

$$\lambda = f(p_{s_i}) \quad (8)$$

λ is smaller if the position in similarity rank is lower. This means a Web service is inappropriate if it cannot provide the required functionality to the users no matter how well it serves. The relationship between searching accuracy and the formula of λ will be identified to extend the search engine prototype in our future work.

3.4. Application Scenarios

Now we discuss in detail how the functional evaluation component operates in different scenarios.

- If only the keywords vector in the functionality part of the user query is defined, the similarity is computed in

Section 3.2 using the keywords vector r^k of the query and the keywords vector K extracted from the descriptions, operation names, and parameter names.

- If the input and output vectors in the functionality part of the user query are defined, the input similarity and output similarity are computed in Section 3.2 using the input/output vector r^{in}/r^{out} of the query and the input/output vector In/Out of an operation. The functional similarity is a combination of input and output similarities.
- If the whole functionality part of a query is available. The functional similarity of an operation is a combination of the above two kinds of similarities, which is computed using R_f and OP_f.

4. Experiments

The aim of the experiments is to study the performance of our approach compared with other approaches (e.g., the one proposed by [11]). We conduct two experiments in Section 4.1 and Section 4.2, respectively. Firstly, we show that the top-k Web services returned by our approach have much more QoS gain than other approaches. Secondly, we demonstrate that our approach can achieve highly relevant results as good as other similarity based service searching approaches even there is no available QoS values.

4.1. Evaluate QoS Recommendation

In this section, we conduct a large-scale real-world experiment to study the QoS performance of the top-k Web services returned by our searching approach.

To obtain real-world WSDL files, we developed a Web crawling engine to crawl WSDL files from different Web resources (e.g., UDDI, Web service portal, and Web service search engine). We obtain totally 3,738 WSDL files from 69 countries. Totally 15,811 operations are contained in these Web services. To measure the non-functional performance of these Web services, 339 distributed computers in 30 countries from Planet-lab [6] are employed to monitor these Web services. The detailed non-functional performance of Web service invocations are recorded by these service users (distributed computer nodes).

In most of the searching scenarios, users tend to look at only the top items of the returned result list. The items in the higher position, especially the first position, is more important than the items in lower positions in the returned result list. To evaluate the qualities of top-k returned results in a result list, we employ the well-known DCG (Discounted Cumulative Gain) [6] approach as performance evaluation metric. DCG value can be calculated by:

[6]http://www.planet-lab.org

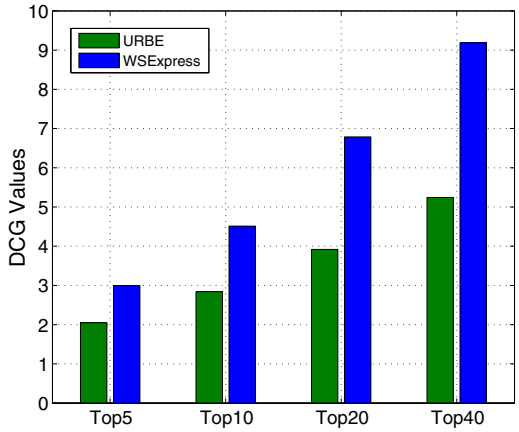

Figure 3. DCG of Top-K Web services

$$DCG_k = \sum \frac{(2^{u_i} - 1)}{\log(1 + p_i)}, \tag{9}$$

where u_i is the i^{th} Web service's QoS utility value and DCG_k is the discounted cumulative gain of top-k QoS utilities in a Web service searching result list. The gain is accumulated starting at the top of the ranking and discounted at lower ranks. A large DCG_k value means high QoS utilities of the top-k returned Web services.

To study the performance of our approach, we compared our WSExpress Web service searching engine with the URBE [11], a keywords matching approach, employing our real-world dataset described above. Totally 5 query domains are studied in this experiment. Each domain contains 4 user queries. Figure 3 shows the DCG values of top-k recommended Web services. The top-k DCG values of our WSExpress engine are considerably higher than URBE (i.e., 2.99 of WSExpress compared with 2.04 of URBE for Top5 and 4.51 of WSExpress compared with 2.84 of URBE for Top10). This means that, given a query, our search engine can recommend high quality Web services in the first positions.

Table 3 shows the DCG values of top-k recommended Web services in the five domains. In most of the queries, DCG values of WSExpress are much higher than URBE. In some search scenarios such as query 2, the DCG values of WSExpress and URBE for Top5 are identical, since in this particular case the most functional appropriate Web services have the most appropriate non-functional properties. In other words, these Top5 Web services have highest QoS utilities and similarity values. However, while more top Web services are considered, such as Top10, the DCG values of WSExpress are becoming much higher than URBE.

Table 3. DCG values (A larger DCG value means a better performance)

Domain	Query ID	Top5		Top10		Top20		Top40	
		URBE	WSExpress	URBE	WSExpress	URBE	WSExpress	URBE	WSExpress
Business	1	0.92319	1.26869	1.28565	2.23302	1.67495	3.54423	2.89918	5.50935
	2	1.78734	1.78734	1.98998	2.77874	2.06252	4.57952	3.71728	6.67234
	3	1.92026	3.92026	2.84943	6.04943	3.11603	8.58338	5.01238	10.14127
	4	2.01145	3.11132	2.12231	3.51222	3.20799	6.50290	6.09782	11.03268
Education	5	3.23724	3.70326	4.50062	5.51494	6.43578	7.92093	8.62007	9.54324
	6	0.57667	2.99091	0.61375	3.20971	2.65626	6.44770	2.92685	10.09865
	7	3.20702	3.20702	4.69254	4.72084	7.07175	7.40789	10.43138	11.23330
	8	1.89318	3.92354	2.81159	5.84194	3.91664	8.03091	4.35944	11.25589
Science	9	2.61991	2.61991	3.18756	3.56422	3.84717	5.71656	4.93499	7.63056
	10	1.87491	3.56752	2.39238	5.08499	3.78722	8.19333	4.81376	11.56507
	11	1.79498	3.79838	2.03920	5.28091	3.77072	8.01607	4.89320	9.98399
	12	4.03468	4.06767	5.28830	5.88298	6.95064	8.00397	7.71733	11.03150
Weather	13	2.96600	3.49303	4.02625	5.63695	5.98159	7.96931	6.62071	9.50699
	14	1.61654	3.61654	3.03344	5.03344	3.34175	7.29602	3.61817	8.74989
	15	2.74210	3.37171	3.42493	5.05464	4.33416	7.45542	4.67509	8.52925
	16	2.69374	3.19009	3.23268	4.91186	5.06458	6.14829	7.67168	7.81805
Media	17	2.75209	3.92562	3.94521	4.09088	4.53422	5.86099	5.98866	7.26564
	18	0.64006	3.07782	1.33133	4.77681	2.09854	5.67896	4.08878	8.57189
	19	0.77538	0.80422	1.30784	1.49103	2.36067	3.41562	2.71115	4.93933
	20	0.90447	2.92768	1.77655	4.51621	2.14091	6.41959	3.05966	8.64248

4.2. Functional Matching Evaluation

In this experiment, we study the relevance of the recommended Web services to the user's query without considering non-functional performance of the Web services. By comparing our approach with URBE, we observe that the top-k Web services in our recommendation list are highly relevant to the user's query even without any available QoS values.

The benchmark adopted for evaluating the performance of our approach is the OWL-S service retrieval test collection OWLS-TC v2 [8]. This collection consists of more than 570 Web services and 1,000 operations covering seven application domains (i.e., education, medical care, food, travel, communication, economy, and weaponry). The benchmark includes WSDL files of the Web services, 32 test queries, and a set of relevant Web services associated to each of the queries. Since the QoS feature is not considered in this experiment, we set the QoS utility value of each Web service as 1.

Top-k recall ($Recall_k$) and *top-k precision* ($Precision_k$) are adopted as metrics to evaluate the performance of different Web search approaches. $Recall_k$ and $Precision_k$ can be calculated by:

$$Recall_k = \frac{|Rel \cap Ret_k|}{|Rel|}, \tag{10}$$

$$Precision_k = \frac{|Rel \cap Ret_k|}{|Ret_k|}, \tag{11}$$

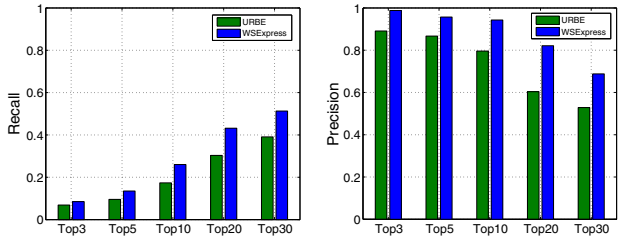

Figure 4. Recall and Precision Performance

where Rel is the relevant set of Web services for a query, and Ret_k is a set of top-k Web services search results.

Since user tends to check only top few Web services in common search scenario, an approach with high top-k precision values is very practical in reality. Figure 4 shows the experimental results of our WSExpress approach and the URBE approach. In Figure 4(a), the top-k recall values of WSExpress are higher than URBE. In Figure 4(b), the top-k precision values of WSExpress are considerably higher than URBE, indicating that more relevant Web services are recommended in high positions by our approach.

5. Related Work

Web service discovery is a fundamental research area in service computing. Several papers can be found on discovering Web services through syntactic or semantic tag match-

ing in a centralized UDDI repository [10, 13]. As discussed before, since UDDI repository is no longer a popular style for publishing Web services, these approaches are not practical now.

Text-based matching approaches have been proposed for querying Web service [4, 14]. These works employ term frequency analysis to perform keywords searching. However, most text descriptions are highly compact, and contain a lot of unrelated information to the Web service functionality. The performances of this approaches are not fine in practice. Plebani et al. [11] extract the information from WSDL files for Web service matching. By comparing with other works [3, 5, 7], it shows better performance in both recall and precision. However, it also dose not consider non-functional qualities of Web services. Our searching approach, on the other hand, take both functional and non-functional features into consideration.

Alrifai et al. [1], Liu et al. [9] and Tao et al. [15] focus on efficiently QoS-driven Web service selection. Their works are all based on the assumption: the Web service candidates which can be select for composition have already been discovered and all meet requesters' functional requirements. As mentioned before, under this assumption these approaches cannot be directly applied into Web service search engine. While our proposed approach employs QoS computation into Web service discovering can address this challenge.

6. Conclusion and Future Work

In this paper we present a novel Web service search engine WSExpress to find the desired Web service. Both functional and non-functional characteristics of Web services are captured in our approach. We provide user three searching styles in the WSExpress to adapt different searching scenarios. A large-scale real-world experiment in distributed environment and a experiment on benchmark OWLS-TC v2 are conducted to study the performance of our search engine prototype. The results show that our approach outperforms related works.

In future work, we will conduct data mining in our dataset to identify for which formulas of λ our search approach can achieve optimized performance. Clustering algorithms for similarity computation will be designed for improving functional accuracy of searching result. Finally, the non-functional evaluation component will be extended to dynamically collect quality information of Web services.

References

[1] M. Alrifai and T. Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *Proc. 18th Intl. Conf. on World Wide Web (WWW'09)*, pages 881–890, 2009.

[2] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *IEEE Internet Computing*, 6(2):86–93, 2002.

[3] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *Proc. 30th Intl. Conf. on Very Large Data Bases (VLDB'04)*, pages 372–383, 2004.

[4] K. Gomadam, A. Ranabahu, M. Nagarajan, A. P. Sheth, and K. Verma. A faceted classification based approach to search and rank web apis. In *Proc. 6th Intl. Conf. on Web Services (ICWS'08)*, pages 177–184, 2008.

[5] Y. Hao, Y. Zhang, and J. Cao. WSXplorer: Searching for desired web services. In *Proc. 19th Intl. Conf. on Advanced Information System Engineering (CaiSE'07)*, pages 173–187, 2007.

[6] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information System*, 20(4):422–446, 2002.

[7] Y. Jianjun, G. Shengmin, S. Hao, Z. Hui, and X. Ke. A kernel based structure matching for web services search. In *Proc. 16th Intl. Conf. on World Wide Web (WWW'07)*, pages 1249–1250, 2007.

[8] M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with owls-mx. In *Proc. 5th Intl. Conf. on Autonomous agents and multiagent systems (AAMAS '06)*, pages 915–922, 2006.

[9] Y. Liu, A. H. Ngu, and L. Z. Zeng. Qos computation and policing in dynamic web service selection. In *Proc. 13th Intl. Conf. on World Wide Web (WWW'04)*, pages 66–73, 2004.

[10] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic matching of web services capabilities. In *Proc. 1st Intl. Semantic Web Conf. (ISWC'02)*, pages 333–347, 2002.

[11] P. Plebani and B. Pernici. Urbe: Web service retrieval based on similarity evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 21(11):1629–1642, 2009.

[12] G. Salton. *The SMART Retrieval System—Experiments in Automatic Document Processing*. Prentice-Hall, Inc., 1971.

[13] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services. *Inf. Technol. and Management*, 6(1):17–39, 2005.

[14] Y. Wang and E. Stroulia. Semantic structure matching for assessing web service similarity. In *Proc. 1st Intl. Conf. on Service Oriented Computing (ICSOC'03)*, pages 194–207, 2003.

[15] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Transactions on Web*, 1(1):6, 2007.

[16] Z. Zheng, H. Ma, M. R. Lyu, and I. King. Wsrec: A collaborative filtering based web service recommender system. In *Proc. 7th Intl. Conf. on Web Services (ICWS'09)*, pages 437–444, 2009.