Efficient Service Retrieval from Service Store using Map Reduce

K.V. Augustine, S.K.V Jayakumar,

Department of Computer Science
Pondicherry University
Puducherry
augustine.k.v@gmail.com

Abstract— The paradigm shift from proprietary standards to Global standards in service computing has made web a user centric environment. With the advent of Web service more and more user are on track to team up and share for their benefits. In a user centric environment the service user namely the consumer, an application developer in our context is oriented towards providing more value added services by means of discovery and composition. Here finding all similar services that matches user demand is desired.. Homogenous or similar services at fine grained level are clustered based on their functional similarity. These are stored in a service store called service aggregator which serves as a repository for the consumer to spot out his interest. Similar service discovery includes search of similar single operation or composite operation. In our approach we propose a storage model that can incorporated functional as well as QoS i.e. non functional characteristics so that more specific user demand can be contented. To reduces the time consumed in searching the store we propose map reduce based searching framework. It not only reduces the time consumed in search but also helps the user to match is demand more precisely.

Keywords – Service Discovery, Similar Service, Service Store, Relational model, QoS, MapReduce

1 Introduction

Service oriented computing has gained momentum with the paradigm shift from proprietary standard to global standard. The advent of web services has further boosted the environment into more user centric were more and more user collaborate for their benefits. The technology support that Web Service [1] provides using XML standards such UDDI, WSDL and SOAP has reduced the challenge in interoperability to data and communication level. This enabled the increase in number of services both in demand and offer. The competitive push of the providers and competence pull of consumers has made this environment an ever green research area.

As the number of services has increased the effort needed by the consumer to run through each service to find the service of his choice is taxing. To overcome this challenging an effective discovery mechanism is significant. The major challenge lies in digging out the optimal service precisely. The basic sources for discovery are UDDI where the description and location of the service is depicted and WSDL [4] where the interface, operations input and output parameters are provided. In traditional approach [5][12][15][17]the search is based on keyword in public UDDI registries, where the keywords in the user query are matched with the keywords in the description. This approach lack precision as it returns irrelevant services and also misses relevant services. Moreover users are willing to be more precise in their request rather than keywords. This challenge has been addressed by providing semantic annotations to describe services using OWL-s [3] or WSDL semantic [14]. But again the ontology based approach suffers from performance problem due to ontology reasoners and WSDL approach suffers from precision problem. Moreover the dynamic nature of the services provider and service consumer has made service discovery an ever improving model.

In some context the user namely a valued added service developer may went through the process of service discovery and ends up in a service which he found inappropriate for some reasons, he may prefer to find similar operations that takes similar inputs/outputs to the ones just considered together with that best suites his preferences. It is reasonable to support such similar services search which assist in discovering similar service and user preferences. Such approach should be efficient for the consumers to find the desired service.

In our approach we would like to propose and efficient discovery methods that can complement with user centric environment where user demand matched to both functional and non functional characteristics of similar operations as well potentially composible operations. The approaches in the paper include

- Similarity measures based on the metadata available from the WSDL description and matching similar datatypes into concepts and algorithms to retrieve the underlying semantic. as in.[16][18]
- We propose a tuple based search model where searching of similar operations can be based on tuple matching using map reduce concepts. The approach can support both the retrieval of single similar operation and composible operations.

The remainder of this paper is organized as follows : Section 2 Literature survey 3 Proposed system 4 Extracting Similar operation section 5 Framework for service search Section 6 Experimental Evaluation Section 7 Conclusion and future direction.

2 Literature Survey

In discovery of similar services two type of services can be identified, single service that matches the user preference may be termed as atomic or elementary service or if a single service does not satisfies his request then set of service termed as composible services that satisfies his preferences.

2.1 Similarity measurement and similar concept clustering

The functionality offered by a service depends on its Operations. So the challenge lies in finding similar operations. The operations are similar if they have similar input and similar output. In case of sequence of operation where the intake of the first operation is the expected input and the outcome of the last operation is the desired output with intermediate operations using the output of its preceding operation. Due to heterogeneity nature in naming operation, input and output for instance "*TravelByDestinatio*n" and "*CabByLocation*" both represent similar type of service with similar operation "BookTicket" and "ReserveTicket". If input parameter to the first operation is "*Location*" and to that of the second operation is "*Area*" which are similar in nature .The challenge lies in

- Finding an appropriate similarity measure ie finding association between similar service, similar operation, similar input and similar output for example "book" and "reserve".
- Clustering similar terms into concepts for example giving common concept name ticket {book, Reserve}.

The similarity measurement approach proposed in [20] is based on scheme matching of the input and output parameters; schema matching has less precision because it does not consider the underlying semantics. In [16] [18] similarity measurement is proposed based on association rules and clustering concepts using agglomeration algorithm. In both the approach the similarity is measured using the measures of support and confidence. In [18] for further improving clustering they employ domain taxonomy. The approach similar to [18] is proposed in our work as it has more precision compared to the other two approaches[20][16]. Multiple sources of evidence for concept clustering which can incorporate domain ontology and web contents will also considered.

2.2 Storage model for simple and composible services

As there exists two types of similar services namely single service or composible service two types of search has to be done. An efficient storage model that can incorporate both the type of searches is expected.

In [18] they proposed a directed graph based search model representing each operation as a node and the composition opportunity as directed edges, and assign the weight of the edge with similarity matching score between input and output but this approach lack scoring based on multiple attributes as the increase in number edges between the nodes is inversely proportional to the efficiency of the search. In [19] tuple based storage model is proposed but it does not include both the types of service ie single and composible. In our approach we would like to propose a relational model that can incorporate both the type of services and also the QoS preferences which is lacking in [18].

2.3 Quality of Service Modeling

QoS of web service are described using QoS description languages [7] which may be an ontology language like DAML QoS [2] or syntactical language like WSOL [13]. QoS model can be classified based on different aspects such as performance, security, stability, user satisfaction [9]. Here in our approach we adopt the QoS aspects based on Performance and Stability quality along with cost and reputation

2.3.1 Performance Quality

Response Time: The time taken to send a request and to receive the response. The Response Time is measured at an actual Web service call and it can be calculated as difference between request completion time and user request time

$$P_{Resp} = R_{comp} - U_{reqs} \quad (1)$$

2.3.2 Stability quality

Successability: Successability is defined as the extent to which Web services yield successful results over request messages. It is the ratio of successfully returned messages after requested tasks are performed without errors.

$$S_{sucs} = \frac{M_{Resp}}{M_{Reqs}} \quad (2)$$

2.3.3 Others factors

Execution Cost: it is termed as the amount of fee the provider charges for utility of his service it can be represented by Q_{cost} cost

Reputation: The value of the reputation is defined as the average ranking given to the service by consumers. It can calculated as the ratio of average of user rank to total number of users

$$Q_{Repu} = \frac{R_{User}}{N_{User}} \quad (3)$$

The qualities described above give the quality measurement metrics from elementary services in case of composite services the quality should be based on the aggregation[3] of the above metrics which is depicted in table II.

3 Proposed System

3.1 Overall framework for service aggregator with definition of terms

In the section we describe the overall framework of our proposed system which is given in Fig1 and brief description o each component in it.

Keyword based crawler engine It is the search engine that extracts the service URL from the repositories based on some keyword matching.

WSDL extractor: Based on the URL's the corresponding WSDL files are extracted from the service provider's site.

I/O similarity matcher: The similarity matcher matches the input the similar based on some IR matching methods like TF/IDF can creates a bag of terms

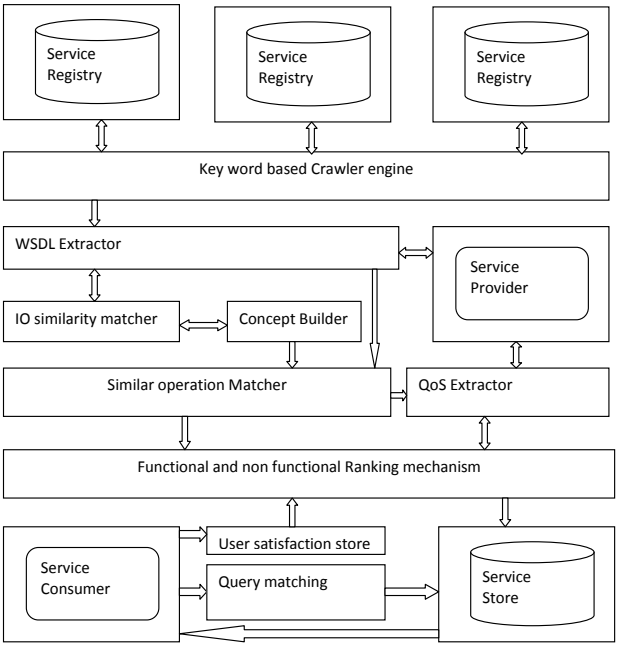

Fig 1: Overall framework for Service aggregator Model

Concept Builder: It is the work of the concept builder to find the association between the terms and clusters them into concepts.

Similar operation matcher : Based on the similarity of the input, output and description of the operation similar operations are clustered together using some clustering techniques.

QoS extractor: The QoS extractor finds the quality of service aspects as provided by the service providers for the service that are clustered

Functional and non functional based ranking : Based on the QoS and similarity measure the service are stored in the service store for retrieval

User satisfaction store: User satisfaction can be graded based on some scale and service store can further refined based on the satisfaction

Query matcher: It is the matcher the matches the user query to find the similar services that or of interest to the user based on the requirement and constrains

3.2 Storage Model of Service aggregator

In this section we propose relational based storage structure that can store both type of operation ie atomic or composible. Here we make a common sense assumption that the number of composible operation should to restricted to some maximum threshold as the increase in number of operation may increase the execution time.

3.3.1 Basic tuple structure

We propose three basic tuple structure namely Similarity Index, Masterpool , QualityStore to be store in the service aggregator storage. The structure of each is detailed below

3.3.2 Structure of SimilarIndex

The SimilarIndex tuple is given by the structure $<S_{mcode}, S_{code}, O_{code}, I_{para}, O_{para}, R_{first}, R_{last}>$, Where S_{mcode} is the code given to similar operation in the MaterPool, S_{code} is the code of the service containing the operation , O_{code} in the operation code, I_{para} is the input parameter to the operation and O_{para} is the output parameter to the operation. R_{first} and R_{last} are the first and last record of $S_{mcode.}$

3.3.3 Structure of MasterPool

The MasterPool tuple is given by the structure $<S_{mcode}, S_{code}, O_{code}, S_{name}, O_{name,} I_{para}, O_{para,} O_{cons}, O_{seq}>$, sorted by S_{mcode} Where S_{mcode} is the code given to similar operation in the mater pool, S_{code} is the code of the service containing the operation , O_{code} in the operation code, I_{para} is the input parameter to the operation, O_{para} is the output parameter to the operation, S_{name} is the service name, O_{name} is the operation name, O_{cons} is construct of the operation atomic or composite and O_{seq} sequence of operation in case of composible operation.

3.3.4 Structure of QualityStore

The QualityStore tuple is given by the structure $< O_{code}, Q_{xmax,} Q_{xmin}>$, where O_{code} is the operation code, $Q_{xmax,}$ Max value of the quality and Q_{xmin} is the minimum value of the quality.

Table 1. Structure of Similar Index

Smcode	Scode	Ocode	Ipara	Opara
SM1	S1	O1	ZipCode	Temperature

Table 2. Structure of quality store

Ocode	QEmax,	QEmin
O1	0.56	0.45

Table3. Structure of Master Pool

Smcode	Scode	Ocode	Sname	Oname	Ipara	Opara	Ocons	Oseq
SM1	S1	O1	Weather forecast	GetTempByZipcode	ZipCode	Temperature	atomic	null
SM1	S2	O2	-	-	Areacode	Warmth	Sequence	O3O4
SM2	S3	O3	Area locator	GetAreabyareacode	Areacode	Areaname	atomic	null
SM3	S4	O4	Climatefind	Getwarmthbyarea	Areaname	Warmth	atomic	null

4 Extracting Similar Operations

In this section we provide the method for similar operation extraction. We adhere to the same approach provided in [18] as it has higher precision when compared to the approaches in [20] and [16]. We propose that the improvement in clustering can be enhances by comparing with multiple source of evidence.

The approach is based on the heuristics that name of the operations, input/output parameters is often combined as a sequence of terms eg. *ZipCodeToTemperture*. Another commonsense heuristics is that the words trends it express the same semantics concepts if they often occur together[20].

4.1 Similarity measurement for Text Description

The textual description of service, operation or input/output is done using the tradition IR techniques of TF/IDF where is the term frequency and is defined as ratio of number of occurrence of the term in the document to the total number of terms in the document.

4.2 Similarity measurement of input/output parameters

The input/output parameters are grouped into terms called the term bag. The association between two terms are measured in terms of two probability measures support and confidence. Support is the number of input/output containing the terms t_i to the total number of input/output terms and The association rules are computed using the A-Prior algorithm

4.3 Clustering association to concepts

Based on the association rule the terms are clustered into concept based on the measure (C_{ij} , S_{ij}) using agglomeration algorithm which is the bottom up version of hierarchical clustering.

The algorithm works as follows, each term is initialized to a cluster. It sorts the association rules in descending order of their confidence and then by support. The terms are arranged in the form of square matrix M where each M_{ij} is the tuple (C_{ij} , S_{ij}). The each step the algorithm selects the highest ranking rule above some threshold δ ie $C_{ij} > \delta$, $S_{ij} > \delta$. The two terms are combined into a cluster and the values are adjusted. The algorithm terminates when no more ranking > δ are available. Finally each cluster is grouped under a concept.

4.4 Fine tuning the clustering of Concepts

The clustering algorithm above is an unsupervised bottom up approach. The clusters may have low cohesion due to the fact that only association rules are considered. In [3] it is proposed that domain taxonomy can be used for further refining the clusters. In their approach matching score was calculated and terms are clustered taking this score together with association for clustering. The domain taxonomy is build using a floksonomy. Here we suggest that domain ontology and more web contents can also be added for building up the domain taxonomy

4.5 Discovering similar operations

Operation are defined by the tuple $O_{op} = <D_{op},D_{in},D_{ou}>$. The similarity of operations O_{op1} and O_{op2} are measured by finding the similarity of each term in the tuple.

To measure the text description of the service S_{ds} and the operation D_{op} we using traditional TF/IDF measure as given in the previous section.

Next the similarity of the input and output are measured by considering the under lying semantics. First the similarity of the description of the input/output names are evaluate using TF/IDF then each term in the input/output are replaced by their corresponding concept and the concepts are compared using TF/IDF measure.

Finally the similarity Score between the two O_{op1} and O_{op2} Score (O_{op1} , O_{op2}) is calculated by multiplying each similarity score ie the similarity score of service description score(S_{ds1}, S_{ds2}), similarity score of operation description score(D_{op1}, D_{op2}), similarity score of inputs score(D_{in1}, D_{in2}) and similarity score of outputs score(D_{ou1}, D_{ou2}) by a weighing factor w_i respectively for each score and then summing them.

w_i are considered in such a way that the sum of w_i is unity. If the Score (O_{op1} , O_{op2}) > ω where ω is the given threshold then the operation O_{op1} & O_{op2} are considered similar.aggregate store

5 Famework for Service search using Map Reduce

In.in this section we provide a framework as given in fig 2 which we propose to use to search similar services for the user demand. in order to improve the efficiency the framework is designed using the concepts of mapreduce[6].

5.1 Mapping user query

Map part

When the user submits the query in the form $Q=<I_{use},O_{use},P_{ex},P_{cost}.....>$, The similarity index table is partitioned into τ parts and each part is assigned to the function

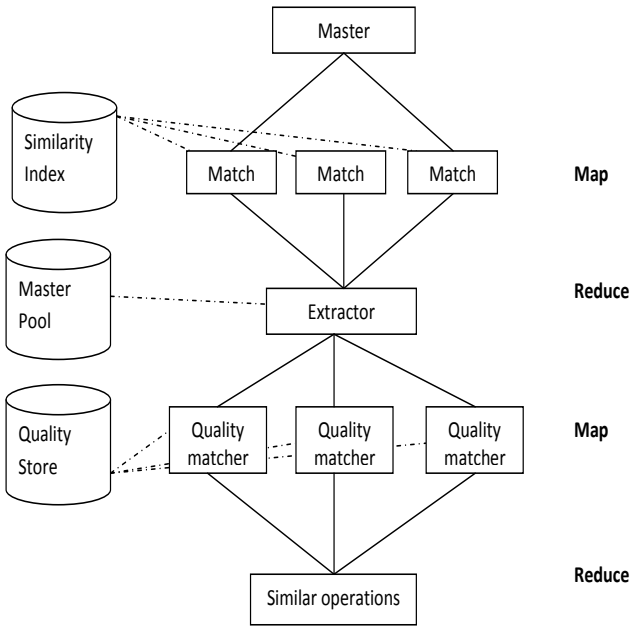

Fig 2: Framework for similar service search

Algorithm 1 :Match for matching user query and index table
Input : I_{use},O_{use}, N_1,N_2,M_{sim}i, SI_{code}i
Output: M_{sim}i, SI_{code}i, R_{first}, R_{last}

1. *Iinitialize* M_{sim}i = 0, SI_{code}i=null
2. **FOR** j various from N_1 to N_2 and each O_{code}j// from similarity index table
3. **COMPUTE** score((I_{use},O_{use}),($I_{para}O_{para}$))
4. **IF** score((I_{use},O_{use}),($I_{para}O_{para}$))> M_{sim}i
5. **THEN** M_{sim}i = score((I_{use},O_{use}),($I_{para}O_{para}$))
6. **ASSIGN** SI_{code}i = S_{mcode}j
7. **END IF**
8. **ENDFOR**
9. **Return** M_{sim}i, SI_{code}i, R_{first}, R_{last}

Fig 3: Matching user query with similarity Index.

Match(I_{use},$O_{use,}$ N_1,N_2,M_{sim}i, SI_{code}i) Where $N_2 - N_1 = \tau$ and i = 1,…., p .

The value of τ is number of records in SimilarIndex divided by number of partitions p. N1 and N2 are first and last record of a particular partition. The output of the function will be the maximum similarity score M_{sim}i, and the similarity code SI_{code}i for that score. The algorithm is given in Fig 3.

Reduce part

In reduce part the output (M_{sim}i, SI_{code}i)for i = 1 to p returned by the Match function is given as an input to Extractor(M_{sim}i, SI_{code}i , O_{code}) which returns the set of operation code O_{code}i (where i = 1 to n) that has maximum SI_{code} value. The algorithm is given in Fig 4.

5.2 Retrieving similar operations

Map part

The operation code return is combined with each user preference and the send to the function Matchquality(O_{code}i, p_J) where i varies from 1 to n and j=1 to m where m is the number of preference. The number of partitions depends on the value of j. The algorithm is given in Fig 5.

The output of the matchquality function is the set of operation the satisfies the user preference for each preference

Reduce part

The set returned by each of the match quality function is sent as the input of the similaroperation. The similaroperation function returns the intersection of all the returned operation by match quality function which is the set of similar operations the matches the user query.

6 Experimental Evaluation

We have implemented our prototype, named as ServicePool, to discover the homogeneous services based on the approach described above. we employ a server that manages the pool in aspects of service clustering, discovery and execution. The server records the services. registered in the pool, including the service URL, service names, operations and inputs/outputs. We present the user interface for service discovery and subscription, as shown in Figure 6. Such UI is so friendly that: (1) the users can do basic operation easily and the discovered results are presented to the user with whole QoS spectrum according to the users' preference. As Operation Name, Input, Output, Description of the operation, URL of the service and the quality details Given the users' request, ServicePool can return the operations with similar inputs and outputs,. Note that the returned results by ServicePool are evaluated by using the usual metrics employed in traditional IR communities, we evaluate our approach with the recall (r) and precision (p).Also we measure the retrivial efficiency of the system from other system as we use simple relational model and Map reduce concept.

Algorithm 2 :Extractor the extract the operation that has maximum similarity score
Input : M_{sim}i, SI_{code}i, p (number of partitions)
Output: set of OR_{code}

1. *Initialize* O_{code}j =null
2. **FOR** i various from 1 to p
3. **ASSIGN** SI_{code} = Max(M_{sim}i, SI_{code}i) // SI_{code}i that has maximum M_{sim}i value
4. **END FOR**
5. **IF** S_{mcode} =, SI_{code}
6. **FOR** j varies from R_{first} to , R_{last} // records for Masterpool
7. OR_{code} = OR_{code} ∪ O_{code}
8. **END FOR**
9. **ENDIF**
10. **RETURN** OR_{code}

Fig 4: Extracting operation from master pool

Algorithm 3 :Match quality for matching the operations with user preferences
Input : OR_{code}, P_i
Output: OP_{code}i

1. *Initialize* OP_{code} =null
2. **FOR** each element in OR_{code}
3. **COMPARE** the value $QE_{max,}$ QE_{min} in quality store with user preference P_i of quality i
4. **IF** match
5. **THEN** Op_{code}i = Op_{code} ∪ O_{code}
6. **END IF**
7. **ENDFO**R
8. **Return** Op_{code}i // ordered by Op_{code}

Fig 5: Matching quality from quality table

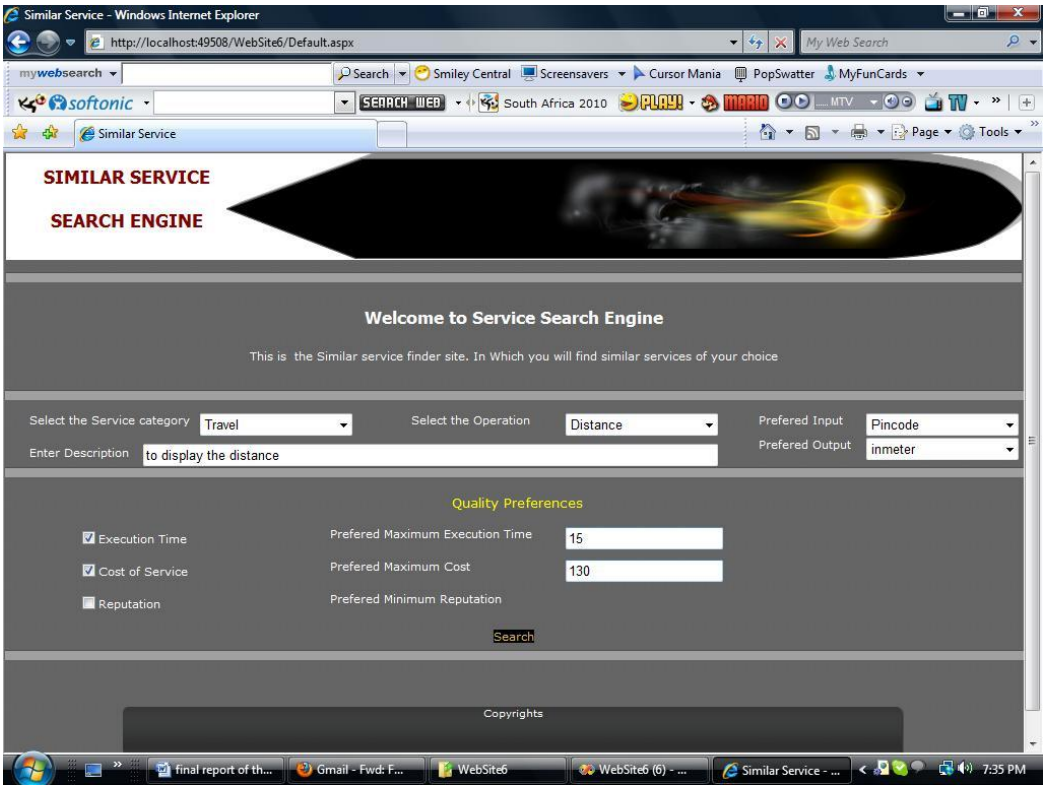

Fig 6: User interface for Service Search from Service Pool

Output of the System is

W1 : **Web Service**: Convertor
Description :Service that coverts various measurement values.
Operation: Curreny Convertor
Descritpion: The converts ruppes to dollars
Input: Rupees
Output: Dollars
Execution Time: 0.23 , **Cost**:$75
URL " http://WWW.convertor.com

W2 : **Web Service**: Convertor
Description :Service that coverts various measurement values.
Operation: Curreny Convertor
Descritpion: The converts Euro to dollars
Input: Euro
Output: Dollars
Execution Time: 0.23 , **Cost**:$75
URL " http://WWW.convertordol.com

7 Conclusion and future Direction

The advent of non proprietary standard, more precisely the proliferation of webservice has made web more usercentric more than a platform for business integration. So service computing has gained momentum and user started using the web more directly for their own benefits. It is evident that orientation towards finding similar services is significant. Here in our approach we have proposed a intermediate storage area called service aggregator which can assist a user, here an application developer to find similar services to the service he have in hand. The WSDL metadata does not provide the semantic of the services it is essential to provide a similarity measurement approach the measure the similarity between different operation based on their description input/output parameters. The similarity measure provided [X.Liu et al. 2009] provides a better precision and recall we adhere to the same approach with little suggestion to improve the domain taxonomy. As for as storage model is concerned we have proposed a relational model which consists of three storage structure similarity index, master pool and quality store. To improve the efficiency we proposed MapReduce based matching and retrieving techniques. In our approach we have considered for single input and single output, the same can be extended for multiple input and multiple output in the future. In our approach we are considering only the WSDL description with the increase in semantic description of the web service using ontology language like OWL_S the similarity measure can be better enhanced resulting in better recall and precision.

REFERENCES

[1]. BV Kumar, S.V. Subrahmanya "Web Service *An introduction*" Tata Mc-Graw-Hill Publishing Company Limited, New Delhi

[2]. C. Zhou, L.-T. Chia, and B.-S. Lee, "DAML-QoS Ontology for WebServices," Proc. Int'l Conf. Web Services (ICWS '04), pp. 472-479,2004.

[3]. D. Martin, "OWL-S: Semantic Markup for Web Services," in Releases of DAML-S / OWL-S, 2004.

[4]. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language.(WSDL) 1.1," 2001.

[5]. Esynaps, http: / /www.esynaps.com, 2009

[6]. J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. InOSDI'04, 6th Symposium on Operating Systems Design and Implementation, Sponsored byUSENIX, in cooperation with ACM SIGOPS, pages 137–150, 2004.

[7]. K.Kritikos abd D.Plexousakis "Mixed interger programming for QoS based web service matchmaking" IEEE transaction on service computing, Vol.2 No.2 April-June.

[8]. Liangzhab Zeng, Boualem Benatallah, "QoS Aware Middleware for Web Servuce composition" IEEE transition of softwre Engineering VOL.30 No5 May 2004

[9]. Quality Model for Web Services v2.0 Committee Draft, September 2005

[10]. S. Robertson, "Understanding Inverse Document Frequency: OnTheoretical Arguments for IDF," J. Documentation, vol. 60, no. 5,pp. 503-520, 2004.

[11]. Selected Resources on Quality of Service (QoS) Specification and Contract-Based Management for XML Web Services 2006

[12]. Strikeiron http: / /www.Stikeiron.com, 2008.

[13]. V. Tosic, B. Pagurek, and K. Patel, "WSOL—A Language for the Formal Specification of Classes of Service for Web Services," Proc. Int'l Conf. Web Services (ICWS '03), pp. 375-381, 2003.

[14]. "Web Service Semantics, "http://www.w3.org/Submission/ WSDL-S/.

[15]. WebServiceX, http: / /www.webservicex.net, 2009.

[16]. Xing Dong, Alon Halvy," Similarity search for web service" Porceeding of the 3oth VLDB conference, Totronto Canda 2004

[17]. Xmethods http: / /www.Xmethods.com, 2009.

[18]. Xuanzhe Liu, Gang Huang, "Discovering homogenous web service community in the user centric web environment", IEEE transition of service computing VOL.2 No2April June 2009

[19]. Xuanzhe Liu, Li Zhou, Gang Huang, Hong Mei "Consumer-Centric Web Services Discovery and Subscription" IEEE International Conference on e-Business Engineering

[20]. Yanan Hao, Yanchim Zhang,"Web service discovery based on schema matching" ACSC conferences Australia 2007