Web Service Management Based on Hadoop

[1]Avinash R. Potdar, [2] Chetan S. Pagar
[1]Assistant Professor, SVIT, Nashik, [2] Lecturer, SVIT, Nashik
Department of Information Technology,
SVIT, Chincholi, Nashik, India

***Abstract* - To solve the disadvantage that huge amount of web service impose on, web service management framework based on Hadoop environment is proposed. In this framework, Hadoop Distributed File System (HDFS) function as a foundation for supporting the HBase and MapReduce. The HBase is assigned to manage the functional and non-functional properties extracted from web service advertisement. To accurately and quickly find the satisfied web service, the HBase table based on request interface is established. In addition, the non-functional property index mechanism based on quality of service (QoS) tree is also designed for further enhancing the performance of non-functional property retrieval. To satisfy the user's complex requirement, the searching optimal solution in all possible paths which depends on MapReduce is proposed. The results of the extensive experiment show that this framework is more suitable for large-scale web service management. The proposed algorithm provides better performance.**

***Index Terms* - Web Service; Web Service Management; Distributed Environment; Hadoop.**

I. INTRODUCTION

With the rapid growth of the internet and World Wide Web, web service is quickly becoming the important part in the service-oriented community. However, the huge amount of web service and the ubiquitous usage of web services to accomplish the complex tasks may trigger the following problems.

The first problem has something to do with the storage efficiency. As more and more customers delegate their task to web service, one can expect a huge web service repository to be fast and efficiently searched. The web service advertisement information is the loose structure, and it's not convenient to be managed by the relational database. Moreover, the traditional central model, such as UDDI [1] platform, cannot easily scale to support a large number of web service.

The other problem leads to the decreasing performance of web service selection for arranging several services into a complex one with QoS guaranteed. Although the artificial intelligence methods are widely used in this field, the searching performance will be decreased as the large number of same or different kind of web services involving with the composition process. In addition, the service selection algorithm based on the global optimum often merely considers the common QoS metrics, like availability, reliability, cost, response time, and reputation. The other non-functional attributes related to different business, such as performance, compensation and so on, are also need to be considered. Hence, the method which considers the aggregated QoS is necessary in composition process.

To address these problems, an efficient web service management framework based on Hadoop is introduced. This framework can overcome the drawback which occurs in the traditional web service management infrastructure. In addition, the two components of Hadoop, HBase and MapReduce, is integrated into this framework. To strengthen the retrieving performance of the functional and non-functional properties, the tables in HBase is designed in terms of the interface-based and QoS tree index. After obtaining the qualified web service, the parallel algorithm based on MapReduce supports us to search all possible solutions and select the optimal solution for satisfying the customer's requirement.

This paper is organized as follows. Section 2 presents the related work. Section 3 is the introduction of applying the Hadoop platform into the web service management. In section 4, we conduct experiments to prove the related results. The last section concludes the paper.

II. RELATED WORK

Web service discovery and composition are the most important topics in the field of service-oriented computing, lots of work have been done. Recently, the topic dealing with the large amount of web service also attracts the researcher's attention.

A. Web Service Discovery

Service discovery is an important issue for semantic web service. The web service discovery based on the input and output interface has been studied [2,3]. To increase the efficiency of web service discovery, [4] proposed a semantic search algorithm in UDDI, The efficient matchmaking is also considered in [5]. Similar to this work, the literature [6] also use the index-based mechanism for web service selection, and extends this to p2p network. To overcome the problem resulted from huge number, the distributed model is applied into this area. Literature [7] distributes web service advertisement in CAN network based on the semantics generated by Latent Semantic Indexing (LSI). Also, work in [8,9] proposes a distributed framework for service discovery. In the non-functional property management, a distributed QoS registry architecture is by Gibelin et al. [10]. [11] proposes a Napster like P2P system to support distributed QoS register, and the replication mechanism is introduced to ensure load-balance of the whole architecture and the query efficiency.

B. Web Service Composition

Composing of the simple web service into a complex one also cannot be ignored in web service research. The algorithms [12-14] improve the composition efficiency based on the tree or graph structure. The underlying rationale is that these methods focus on

avoiding the unrestricted growth of the tree depth. In the case of parallel composition, literature[15] first adopts parallel strategy for web service composition to the best of my knowledge.[16] proposes a parallel web service composition framework which partition the composition algorithm based on literatrure[12]into different processor. In addition, the efficient web service composition with QoS guaranteed often considers to combine with artificial intelligence methods[17,18].
Unlike the traditional distributed systems for web service management, our framework based on Hadoop platform can unseamly integrate with discovery and selection mechanism, and is more suitable for managing the loose-structure information of web service. Especially, easily scale to the lager-scale web service data.

III. WEB SERVICE MANAGEMENT FRAMEWORK BASED ON HADOOP

The web service management framework based on Hadoop consists of four layers: web service management, web service property registry, web service composition designer, web service interface. The framework is depicted as follows:

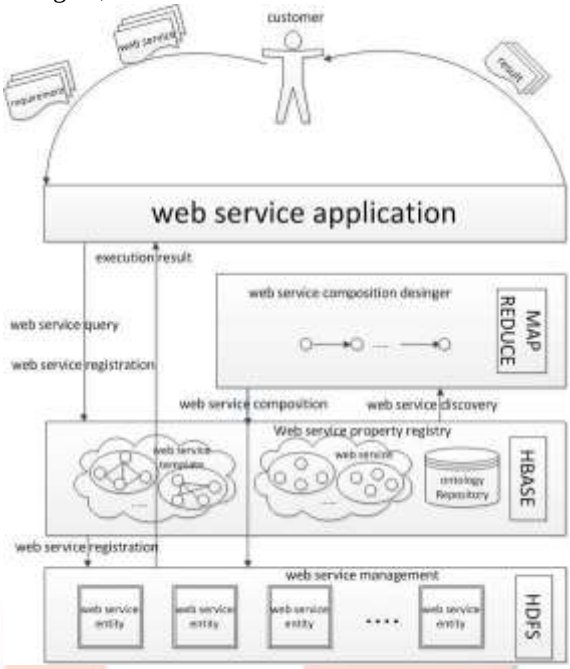

Figure 1: Web Service Management Framework Based on Hadoop

Web service interface layer. It provides an interface to present the data and metadata about web service, which is convenient for customer to interact with this system.
Web service composition designer layer. The main purpose of this layer is to provide customer an optimal solution for their complex requirement. MapReduce is the main component in this layer. The processes which MapReduce can participate into include retrieving the non-functional and functional interfaces which satisfy the customer's requirement from HBase, concurrently select the best matched web services among them and filter the unqualified-QoS web service.
Web service property registration layer. Except that the functional and non-functional properties are stored by HBase, it also maintains the functional and non-functional index, the ontology tree and the QoS tree, which both contribute to enhance the HBase performance. The characteristic of HBase is suitable for the loose-structure data, and a data row has a sortable key and the varying columns. Thus, it can effectively manage the web services with different properties.
Web service management layer. This layer consists of the HDFS. The characteristic of this distributed environment is a high scalability and reliability, where it ensures the correct execution of web service. Besides the executable web services entities, it also manages the optimization algorithms, and data for execution.

A. Web Service Storage Based on HBase

HBase is a robust, scalable and distributed repository which is capable of hosting billions of the web service's advertisement information, thus it plays an important role in web service discovery and selection. However, an efficient index strategy is also significant important for processing large amount of web service. The following section introduces the information retrieving based on index mechanism in HBASE.

1) The non-functional property storage: The QoS requirement is the necessary part in the service selection, however, there is no mechanism for HBase to support the complex query, which often appears in the non-functional property index. For example, the customer requires the metrics of reliability, availability and reputation is no less than 50. In addition, the web service in different domain demands for the different QoS requirement, thus, the local QoS optimum cannot be ignored. To strengthen the HBase, we introduce the QoS tree index. Before we construct QoS tree, we intend to quantify the QoS metrics and map them into normalized space. Actually, there are three types of relationship among quantified QoS: strong dominance, dominance and ε-weak dominance.

Definition 1. Strong Dominance: If $QoS_1 = (q_1, q_2 \quad q_m)$ dominate $QoS_2 = (q_1', q_2' \quad q_m' \quad)$. $QoS_1 \quad QoS_2$ .If only if

$$\forall i \in \{1, \quad , m\} q_i > q_i'$$

Definition 2. Dominance: If $QoS_1 = (q_1, q_2 \quad q_m)$ dominate $QoS_2 = (q_1', q_2' \quad q_m' \quad)$. $QoS_1 \quad QoS_2$. If only if:

$$\exists i \in \{1, \quad , m\} q_i = q_i'$$

Obviously, the strong dominance is the exception of the dominance, we use the mark $_{all}$ to include this two kinds of condition.

Definition 3. ε -weak dominance: If $QoS_1 = (q_1, q_2 \quad q_m)$ ε -weakly dominate $QoS_2 = (q_1', q_2' \quad q_m')$. $QoS_1 \quad QoS_2$ .If only if:

$\exists i \in \{1, \ , m\} q_i \geq q_i'$, where the number of $q_i \geq q$ is ε and $\varepsilon < m$

According to these relationships, the architecture of QoS tree framework is described as follows:

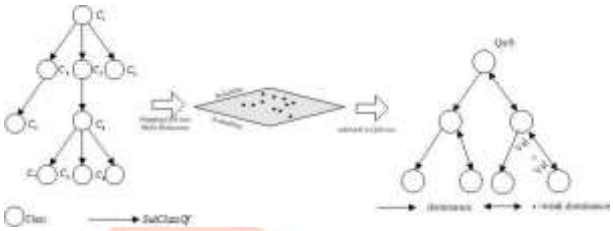

Figure 2. QoS Tree

In the $node_{QoS}$ structure, the $node_{QoS}$ ' managed by *LeftChild* is strongly dominate or dominate by the $node_{QoS}$, and the $node_{QoS}$ ' managed by *RightChild* ε -weakly dominate

by the $node_{QoS}$ .Thus, the relation between parent and its child depends on the node is left child or right child. QoS denotes the non-functional vector (q_1 , q_2 q_n). Val is the average value of QoS vector, $\frac{1}{n} \sum_{i=1}^{n} q_i$.

According to the QoS node's definition, the left and right child can be considered as index node to reduce the searching cost due to the transitive relation. To simplify the index, we construct a virtual index, which is diagonal of the hypercube of the quantified QoS dimension, as the main index. That is. (0, 0) (1, 1) (100,100) is introduced. The detailed algorithm of join and locating the qualified QoS can refer the literature [19]. Therefore, the HBase table in terms of the QoS tree can be designed as follow:

TABLE 1: QoS INDEX TABLE

Row Key	**Time Stamp**	**Relationship**		**Web Service**	
travel90151	T1	Qos	QoS metrics	Serv1	DomName
		Parent	QoSId	Serv2	DomName
		Lchild	QoSId	Serv3	DomName
		Rchild	QoSId	Serv4	DomName
		Parent inbst	QoSId	Serv5	DomName

The Row Key in HBase is the combination of domain and the encoded QoS id, such as (travel90151), where travel represents the web service's domain and 90151 contains the main index value 95 and the sequence number in this main index 151. By using the Row Key, we can concurrently retrieve the qualified QoS in different domain to improve the retrieving performance. The column family Relationship describes the QoSId's relationship in QoS tree. In addition, the leftmost node and the rightmost node can be constructed as a binary search tree for reducing the time cost. Thus, the relationship column for the index nodes stores immediate family member in the binary search tree. The column family Web Service includes url of web service in terms of the QoS id appearing in relationship column.

2) The Functional Property Storage: Besides the non-functional property management, the functional property can also be stored in the HBase. There are two types of table for managing the functional property, ontology and interface table.

In fact, ontology is modeling tool for concept *C*, an ontology can be also represented graphically a tree structure. According to the tree structure, we can conclude that there are four different scenarios in the interface matching process, where *I* denotes request interface and *O* denotes the response interface, need to be considered.1) *O* and *I*'s concepts are the same, Sim(*I* = *O*). 2) The concept *I* subsumes concept *O*, Sim(*I* < *O*). 3) The concept *O* subsumes concept *I* , Sim(*I* > *O*) . 4) The concept *O* is not directly related to concept *I*, Sim($I \neq O$). The degree of the similarity Sim(*I*, *O*) decreases from the scenario 1) to 4), thus, we

can use a generic criterion for accessing the similarity degree between interface *I* and interface *O*.

$$Similarity(I,O) = \max(\{C \mid C \in C_I, C \mid C \in C_O\}) \qquad (1)$$

The ontology table structure is similar to the QoS index table. The column family Relationship includes the concept of immediate family in ontology tree. In addition, it includes the other two types of column family, the Request Interface and the Response Interface, both of them store the interface instance belonging to this concept in Row Key. The table structure is described as follow:

TABLE 2: ONTOLOGY TABLE

Row Key	Time Stamp	Request Interface		Response Interface		Relationship	
conp	T1	Req1	Inst1	Res1	Inst1	Parent	Conp1
		Req2	Inst2	Res2	Inst2	Child	Conp2
		Req3	Inst3	Res3	Inst3	Brother	Conp3

Besides the ontology table, another table is designed for managing the interface relationship. Apparently, the basic method for finding matched web service relies on the interface. Therefore, we create an interface table to search the matched interface and avoid the time cost of irreverent interface comparison. The interface table can be designed as follow:

TABLE 3: INTERFACE TABLE

Row Key	Time Stamp	Web Service	Matched Interface	
conpInst	T1	DomName	Instance	Inst1
			Similarity	0.5
			Service Name	DomName
			Other Interface	conpInst

The Row Key is constructed by the combination of concept and interface id, which helps to concurrently retrieve interfaces belonging to the same concept. The column family Web Service maintains the combination of web service name and the domain ontology which it belongs to. By matching column family Web Service in QoS index table, we can filter QoS-unqualified web service. In Matched Interface column, there are four types of metric to describe the matching information, including instance, similarity, web service name and other interface. The other interface stores the remaining interfaces belonging to the web service producing the instance of response interface in Matched Interface column. The interface table stores the matching information about interfaces. However, the matching relationship between web services should also be maintained.

TABLE 4: WEB SERVICE TABLE

TABLE IV. WEB SERVICE TABLE					
Row Key	Time Stamp	web service		Matched Web Service	
domName	T1	url		WS	domName
				Matched Interface	conpInsts
				Remain Request Interface	conpInsts
				Remain Response Interface	conpInsts

In web service table, column family Matched Web Service maintains the matched web service name, the matched interfaces between them and remaining interfaces, including request and response. The remain interfaces help to find matched interface from the interface table.

B. Web Service Selection Based on MapReduce

By using the HBase, we can efficiently and reliably retrieve the necessary information about web service selection from the web service repository. There are two types of service selection algorithm: simple and complex web service selection.

1) Simple Selection: After fetching all matched interfaces, the simple selection is to find a web service to satisfy the requirement. The algorithm reads the interfaces which match the customer's requirement from the Interface table, and select the web service who matches all interface requirements, thus, the algorithm of selecting the best matched web service (BMWS) is listed as follows:

Algorithm: *BMWS*
Input: *Interface requirement*
Output: *best matched web service*

```
1. Function Mapper(key,value)
2.     Ouput(wsn, matched Interface);
3.  EndMapper;
4. Function Reducer(wsn, matched Interfaces)
5.   For(intfs: matched Interfaces));
```

```
6.          For(rintfs:require Interfaces)
7.              getBestMatchedCombination( );
8.       Output(Null, matchedResult);
9.     EndReducer
```

According to the parallel BMWS algorithm, the value in Mapper function is the combination of the customer's interface requirement, matched interfaces, web service name(*wsn*) and similarity(*I*,*O*). To concurrently match web services, we use web service name as the Reducer key. The value in Reducer is matched interfaces set of *wsn* collected by a Reducer. In each Reducer, we run the web service matching algorithm to find the best match among two web services. The function *getBestMatchedCombination*( ) is to find the optimal interface matching result among interfaces. The key/value of Reducer returns to the final interface similarity results.

2) Complex Selection: To accomplish the complex task, each simple web service must be composed together with QoS reaching the global optimization. The following figure presents us the composition procedure.

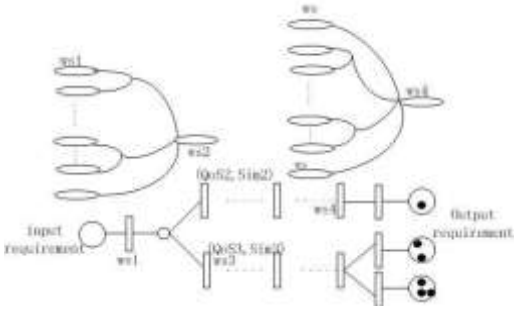

Figure 3. Web Service Composition Graph

We use the Petri Net[20] to describe the web service composition graph, the token in this graph denotes the customer's output requirement. The output requirement can be considered as the start point of the composition process. Any web service which produces one or more this output requirement can be added into the search path. Unless obtain web services where input requirement is satisfied by them, the search process continues. The weight on the edges between web services can be measured by two metrics, similarity and QoS. The evaluation method of composite web service (CWS) is defined as follow:

$$score(cws) = (QoS_{all}, Sim_{all}) = \omega_1 QoS_{all} + \omega_2 Sim_{all} \quad (2)$$

The QoS_{all} evaluation in the following model, such as consequence and concurrent model, can be defined as $\sum_{i=1}^{n} QoS(ws_i)$, $\min(QoS_i)$ irrespectively. The Sim_{all} can be represented by $\frac{1}{n}\sum_{i=1}^{n-1} Sim(ws_i, ws_{i+1})$.

According to the composite web service graph, the optimal solution selection method can be considered as the single-source shortest path(SSSP) algorithm. To reduce the time cost, we should divide the workload of searching process to different computers, thus, the parallel web service selection algorithm decomposes the single path search process to several ones by selecting those web services which satisfy the output requirement as the new start point. The multi-source search process(MSSP) based on MapReduce describes as follow.

Algorithm: *MSSP*
Input: *Interface Table*
Output: *Optimal Solution*

```
1.  Job1.
2.  Function forwardSearch()
3.      findNext();
4.      If(output requirement is satisfied)
5.          Return;
6.       else
7.          forwardSearch();
8.  Job2.
9.  Function Mapper ( key , val )
10.     Ouput(path id, wss);
11.   EndMapper;
12.  Function Reducer(path id, wss)
13.    While(wss.hasNext()){
14.      If (wss.next()==pathid)
15.      goalList.add(wss.outputInterface);
16.    }
17.    Regression(goalList, wss);
```

```
18.    getFinalScore();
19.    Ouput(Null, score);
20.    EndReducer
```

The MSSP algorithm consists of two jobs. Job1 is assigned to collect the web services which belong to the specified search path. The function *findNext*() is to find the web service which can be executed in terms of the output list, the output list is updated when it obtains one. The recursive invocation stops until the output requirement is satisfied. After that, the path id, web service name belonging to this path and its interfaces are written in a file and stored in HDFS.

Job 2 reads this file to start the parallel search process. The output key of Mapper function is the path id, which is denoted by web service name of the start point, and the value is web services(wss) belongs to this search path. In the Reducer function, *goalList()* collects the input interfaces of optimal matched web service. *Regression()* recursively searches the matched web services until to satisfy the input requirement. The output value of Reducer function is the score of this solution.

IV PERFORMANCE EVALUATION

We have implemented the all these algorithms based on the Hadoop platform. For these experiments, we use a distributed cluster with cpu 2 GHz, RAM = 8 GB, Linux operating system. All algorithms have been implemented in Java version 1.6. We randomly generated composition requests and report the average time of several runs by using the dataset from generator of WS-Challenge 2008 and WSBen[21].

Figure 4. The QoS Index and Interface index Cost

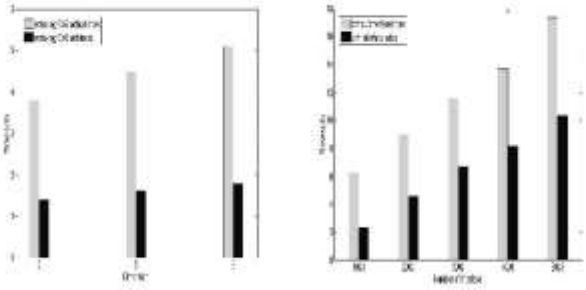

Figure 4 shows us the query cost when we retrieve the functional and non-functional properties from HBase with or without using index mechanism. From the figure4 (a), The QoS[22] query cost maintains within a low level since the strong-dominance or dominance relation in the QoS tree contributes to reduce the unnecessary comparison. In addition, the time consumption doesn't grow with the QoS dimension increasing since the comparison times in QoS tree doesn't grow with the QoS dimension increasing. On the contrary, the time cost increases as the dimension increases when retrieving the QoS without using index mechanism. Therefore, the QoS index tree contributes to reduce time consumption when retrieving QoS from HBase. Figure4 (b) shows us the time consumption comparison of retrieving the matched interface between HBase and the WSDL files generated by WSBen. The latter will pay more IO cost of reading files and repeatedly searching files for finding matched interfaces, while the interface-based index merely produce more storage cost. The experiment results demonstrate that interface-based index is faster than that merely relying on WSDL files.

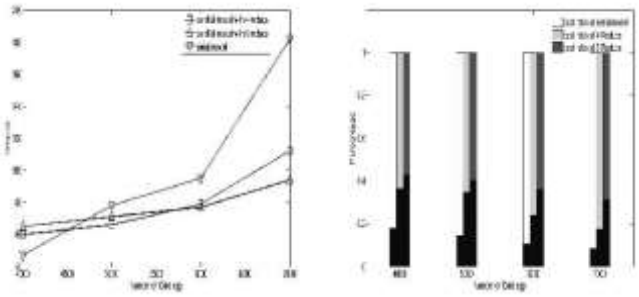

Figure 5. The Time Consumption of Best Matched

In this experiment, we generate a test set which includes 2000 web services with different ontology number by using generator in ws-challenge 2008. As the number of ontology increases, the interface number increases from 1257 to 2187, and the edge number among the web services reaches to 64926, thus, the number of a web service's neighbor rise. We investigate our algorithm's computation time improvement for finding the best matched among the large-scale web service. According to the figure 5(a), the time consumption grows relatively slowly in parallel model as the increase in ontology number, and the serial model reach to 180(s) when the web services with 7000 ontology. Figure 5(b) reflects that the IO cost during the whole process, the additional IO cost results from the higher communication and data movement in cluster, especially, with more reduce number involving the computing. While the performance can be further improved as the more computers involve in this process though it brings more extra IO cost.

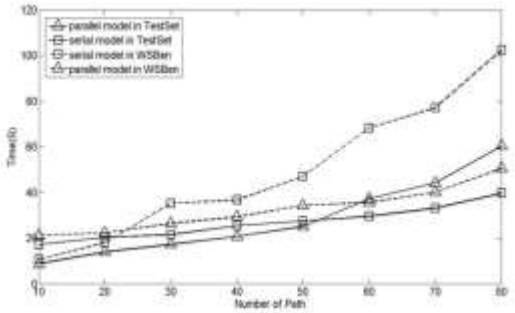

Figure 6. Composition Time Consumption

We investigate our algorithm's computation time improvement in these two types of dataset for finding the optimal solution with different number of path to the web service satisfying the request requirement. In this experiment, we add more response interfaces to the user's requirement to get more output combination. According to the results, the serial model still outperforms the parallel algorithm when applying into this algorithm to a few numbers of paths. Although the parallel algorithm distributes the searching process on each computer, the cost of data movement and communication overpass the computing consumption due to the MapReduce mechanism. As the number of path and the longest shortest path of the web service network increase, the iterative search of the best solution decreases the performance of serial process, while the parallel model has the apparent improvement.

V CONCLUSION

There is a plethora of web service management and selection based on distributed environment, none of them considers the management efficiency when facing the huge amount of web services, thus, a web service management framework based on Hadoop is proposed. To improve the performance, we introduce Hadoop into this area, where HBase manages the functional and non-functional properties and MapReduce is assign to parallel handle the composition related tasks. The experiments show that the framework has good performance in retrieving process with the help of index mechanism. In web service selection experiments, the performance is far better than that of the serial method as the solution path become complex. We must admit that the IO cost of the MapReduce mechanism will decrease its performance in small-scale web service processing. However, a series of experiments prove that this framework is more adaptive for managing large-scale web service data.

Our future work is to research about the ETL, data mining , statistic analysis and complex network algorithms, and integrate them with the Hadoop enviroment for better managing web services and further improving web service selection efficiency.

REFERENCES

[1] T. Bellwood et al. Universal Description, Discovery and Integration specification (UDDI) 3.0. http://uddi.org/pubs/uddi-v3.00-published-20020719.htm
[2] Li, L., Horrocks, I. “A Software Framework for Matchmaking based on Semantic Web Technology,” In: proceedings of the 12th International Conference on World Wide Web, 2003, pp. 331-339.
[3] Srinivasan, N., Paolucci, M., Sycara, K.P., “An Efficient Algorithm for OWL-S Based Semantic Search in UDDI,” In: First International Workshop on. Semantic Web Services and Web Process Composition., 2004, pp. 96-110.
[4] Bellur, U., Kulkarni, R., “Improved Matchmaking Algorithm for Semantic Web Services Based on Bipartite Graph Matching,” In: IEEE International Conference on Web Service, 2007, pp. 86-93.
[5] Constantinescu, I., Binder, W., “Faltings, B.: Flexible and Efficient Matchmaking and Ranking in Service Directories,” In: IEEE International Conference on Web Service, 2005, pp.5-12.
[6] Skoutas D., Sacharidis D. Kantere V. and Sellis T., “Efficient Semantic Web Service Discovery in Centralized and P2P Environments,” In Proceedings of International Semantic Web Conference, 2008, pp. 583-598.
[7] C. Tang, Z. Xu, and S. Dwarkadas, “Peer-to-peer Information Retrieval using Self-organizing Semantic Overlay Networks,” In Proceedings of pecial Interest Group on Data Communication, 2003, pp. 175-186.
[8] F. Emekci, O. D. Sahin, D. Agrawal, and A. E. Abbadi., “A Peer-to-Peer Framework for Web Service Discovery with Ranking,” In: Proceedings of the IEEE International Conference on Web Services, 2004, pp. 192-199.
[9] F. B. Kashani, C. C. Chen, and C. Shahabi., “WSPDS:Web Services Peer-to-peer Discovery Service,” In: Proceedings of International Symposium on Web Servicesand Applications, 2004, pp. 733-743.
[10] N.Gibelin and M.Makpangou, “Efficient and transparent web-services selection,” Proceedings of the 5th International Conference on Service Oriented Computing, 2005, pp. 527-532.
[11] Fei Li, Fangchun Yang, Kai Shuang. “Q-peer: A Decentralized QoS Registry Architecture for Web Services,” Proceedings of the 5th International Conference on Service Oriented Computing, 2007, pp. 145-156.
[12] A. Zhou, S. Huang, and X. Wang, “BITS: A Binary Tree Based Web Service Composition System,” Int. Journal of Web Services Research, vol. 4, 2007.
[13] H. Tang, F. Zhong, and C. Yang, “A Tree-Based Method of Web Service Composition,” Proc. of the Int. Conf. on Web Services , 2008, pp.768-770.
[14] M.M. Shiaa, J.O. Fladmark, and B. Thiell, “An Incremental Graph-Based Approach to Automatic Service Composition,” Proc. of the Int. Conf. on Services Computing, 2008, pp. 148-165.
[15] J. Pathak, S. Basu, R. Lutz, and V. Honavar, “Parallel Web Service Composition in MoSCoE: A Choreography-Based Approach,” Proc. of the 4th European Conf. on Web Services, 2006, pp. 3-12.
[16] Hennig P., Balke W., “Highly Scalable Web Service Composition Using Binary Tree-based Parallelization,” IEEE

International Conference on Web Services, 2010, pp. 123-130.
[17] Yue Ma, Chengwen Zhang, "Quick Convergence of Genetic Algorithm for Qos-Driven Web Service Selection," Computer Networks,2008,52(5):1093-1104.
[18] G. Canfora, M. Di Penta, R. Esposito, and M. L.Villani. "An Approach for Qos-Aware Service Composition Based on Genetic Algorithms," In: Proceedings of the 2005 conference on Genetic and evolutionary computation, 2005, pp. 1069-1075
[19] XiLu Zhu, Bai Wang., "A Distributed Quality of Service Index Framework," In : The IEEE Asia-Pacific Services Computing Conference, 2010, pp. 23-130.
[20] David Liu, Kincho H. Law, Gio Wiederhold. "Analysis of Integration Models for Service Composition," Proceedings of the 3rd international workshop on Software and performance, 2002, pp. 158–165.
[21] Seog-Chan Oh, Hyunyoung Kil, Dongwon Lee, et al. "WSBen: A Web Services Discovery and Composition Benchmark," In Proc. of the fourth International IEEE. Conference on Web Service, 2006, pp.239-248.
[22] E. Al-Masri and Q. H. Mahmoud, "Investigating Web Services on The World Wide Web," Proceeding of the 17th International Conference on World Wide Web, 2008, pp. 795-804