Efficient semantic web service matching using common ontology

Cao Jiuxin[1,2] Qin Yi[1,2] Zhang Song[3] Liu Bo[1,2] Dong Fang[1,2]

([1]School of Computer Science and Engineering, Southeast University, Nanjing 211189, China)
([2]Key Laboratory of Computer Network and Information Integration of Ministry of Education, Southeast University, Nanjing 211189, China)
([3]Nanjing Normal University, Nanjing 210042, China)

Abstract: To solve the bottleneck problem in centralized service discovery methods, a novel architecture based on domain ontology for semantic service discovery is proposed. This distributed architecture can adjust the domain partition and allocate system resources automatically. The characteristics of this mechanism are analyzed, including scalability, self-organization and adaptability. In this mechanism, semantic web service discovery is separated into two parts. First, under balance tree topology, registry proxy can rapidly forward requests to the objective registry center, and avoid the bottleneck problem. Secondly, a semantic distance based service matching algorithm is proposed to promote the effect of service searching. The results of simulation experiments show that the proposed mechanism can serve as a scalable solution for semantic web service publication and discovery. And the improved matching algorithm has higher recall and precision than other algorithms.

Key words: service discovery; domain ontology; semantic web service; semantic distance
doi: 10.3969/j. issn. 1003 - 7985. 2012. 03. 007

Recently, with the development of web service standards and technology, a great quantity of web services has been provided on the Internet. Due to the explosion of service information, how to rapidly and precisely find the user required service in such a large online group has become a major problem in the web service system[1]. Web service discovery technologies as the solution to this problem have been studied by researchers for years.

Received 2012-02-21.
Biography: Cao Jiuxin (1967—), male, doctor, professor, jx. cao@seu. edu. cn.
Foundation items: The National Basic Research Program of China (973 Program) (No. 2010CB328104, 2009CB320501), the National Natural Science Foundation of China (No. 61070161, 61070158, 61003257, 61003311), the National Key Technology R&D Program during the 11th Five-Year Plan Period (No. 2010BAI88B03), the Foundation of Jiangsu Provincial Key Laboratory of Network and Information Security (No. BM2003201), Open Research Fund from Key Laboratory of Computer Network and Information Integration of Ministry of Education (Southeast University).
Citation: Cao Jiuxin, Qin Yi, Zhang Song, et al. Efficient semantic web service matching using common ontology[J]. Journal of Southeast University (English Edition), 2012, 28(3): 292 - 298. [doi: 10. 3969/j. issn. 1003 - 7985. 2012. 03. 007]

In most current service discovery methods, WSDL[2] (web service description language) is used to describe the interface of web services. However, without semantic description of service functionality, knowledge reasoning is disabled in these methods, which lowers the precision of service discovery. On the other hand, UDDI[3] (universal description discovery and integration) as a current service discovery standard is adopted to support the keyword-based service matching mechanism. But, this framework has two limitations[4].

First, centralized service registries based on UDDI may easily suffer from problems such as the performance bottleneck of vulnerability to failures with the increase in the number of service consumers and requests. This inherent disadvantage prevents web services from being applied in large scalable service networks. To overcome this problem, the decentralized architectures, such as the P2P network, are designed to enable a scalable web service discovery mechanism. In these frameworks, registry centers separately store services in different domains by using service classification information in the common knowledge database. However, one major problem is that these centers' classifications is predefined and cannot be changed according to the practical situation.

Secondly, the search mechanism of UDDI is based on the key words matching, without any formal semantic description of their capabilities. The lack of any machine interpretable semantics requires human intervention for the service discovery, and reduces the accuracy of service matching[5]. Thus, semantic web service discovery has been proposed based on the semantic web and web service technological concepts. Burstein et al. [6] summarized the abstractions necessary for architecture to support the semantic web service, and indicated the phases of semantic web service interaction. The key idea is to present the functionality of a web service explicitly by using the semantic annotations[7].

In this paper, a novel distributed service discovery architecture is presented, which is organized in a tree form topology. It promotes the efficiency of service registry and discovery with higher adaptability than other methods. Meanwhile, OWL-S[7], as a widely applied standard, is adopted to provide a framework describing both the functions and the advertisements for web services. Because of its capability of abundant semantic information, OWL-S is used to describe the web service advertisements and service queries semantically in our method (UDDI-based service query can easily be transformed to semantic query messages.). Registry centers are classified based on the related OWL-S ontology. Also, our solution relies on registry proxies which are designed to manage registry centers.

For service matching, an improved semantic matching algorithm is proposed. We leverage the method to measure the semantic similarity between the concepts in the service query and the service advertisement, and it is more efficient than other methods. In the processing of ontological concepts matching, we not only consider inheritance (subClassOf) relations, but also the disjoint relationship between concepts. With the weight assignment for different relationships in the ontology graph, the proposed matching approach outperforms the other two algorithms[8-9] in precision and recall according to the experiments.

1 Related Work

Since scalability has become a significant property for web service discovery[10], the distributed service discovery mechanism is introduced for large scale web service applications. He et al.[11] proposed a P2P-based decentralized service discovery approach named Chord4S. It utilized distributed description data of functionally-equivalent services and inspected the capabilities of the popular chord ring to provide service distribution and discovery in a decentralized environment. In our previous works presented in Ref. [12], the distributed web service architecture ensures the effectiveness and scalability of web service systems. However, services with the same functionality are inaccessible when a single failure occurs in this architecture, so we design a new failure tolerance method to improve the robustness of TSWS.

On the aspect of web service matching, methods are classified in two types: one is UDDI-based key words matching technology, and the other is semantic service matching technology. The lack of semantic description information greatly reduces the precision and recall of service matching. Therefore, researchers have proposed many mechanisms in semantic service matching, most of which are based on the similarity of each concept in an ontology graph. In Ref. [8], based on the hyponymy of concepts corresponding to the input and output of web services, semantic matches are divided into exact, plug-in, subsumes, and fail. Exact matches are preferred to plug-in matches, which in turn are preferred to subsume matches. This is a classic and widely cited semantic web service matching algorithm, but it cannot allow for more specific matching in the same degree, which makes it not suitable to the situations of many web services in a network. Therefore, Fu et al.[9] proposed a formula to measure the semantic distance between services through concepts specified in the ontology. Four structural attributes — Path Length, Depth, Local Density and Number Of Down Direction in ontology graph — are considered to improve the matching performance. While in Ref. [13], the depth of concept becomes a factor in the weight assignment function. When the whole edges are weighted, the node rooting table is generated through breadth traversal operation on the ontological concept graph, and the distance of concepts can be calculated for further semantic similarity computation.

In most existing researches, relationships except "subClassof" are rarely considered. In fact, relationships between concepts may have more types, such as "disjoint" or "hasSome". These relationships provide more human intuition similarity between concepts, which help to enhance the precision and recall of algorithms. Therefore, four relationships (subClassOf, instanceOf, hasSome, disjoint) are taken into account as influencing factors during weight assignment to improve the performance of semantic web service matching.

2 Architecture

In this section, we design an effective failure tolerant method for our previously proposed architecture for service discovery to protect our system functionality from being unavailable.

2.1 Tree-form topology

In this paper, the OWL ontology is used as the semantic metadata to identify registry centers. Concepts (domains) in the domain-specific ontology are organized as a directed acyclic graph (tree structure at most of the time) through the relationship "subClassof". Thus our architecture has a tree-form topology called tree for semantic web service (TSWS), as shown in Fig. 1. Four participators are involved, namely ServiceRequester, ServiceProvider, ServiceDiscoverySystem, and domain-specific ontology. The part of the domain-specific ontology acts as a common knowledge supply ontology information for ServiceDiscoverySystem. The whole procedure of service registry and discovery is listed in Fig. 1.

Two types of nodes exist in our main topology.

1) Registry proxy (RP) As managers in the proposed architecture, RP nodes have two major functions: maintaining the structure of the topology and forwarding registration/discovery messages to the target storage nodes. Each RP is related with a concept set in which the concepts are generated from the set of its child node. A route table — neighbor table — of the RP node contains information of its father and children. Therefore, the RP node has the ability of "self-replication", namely a new RP node will be created when new information is added to the neighbor table.

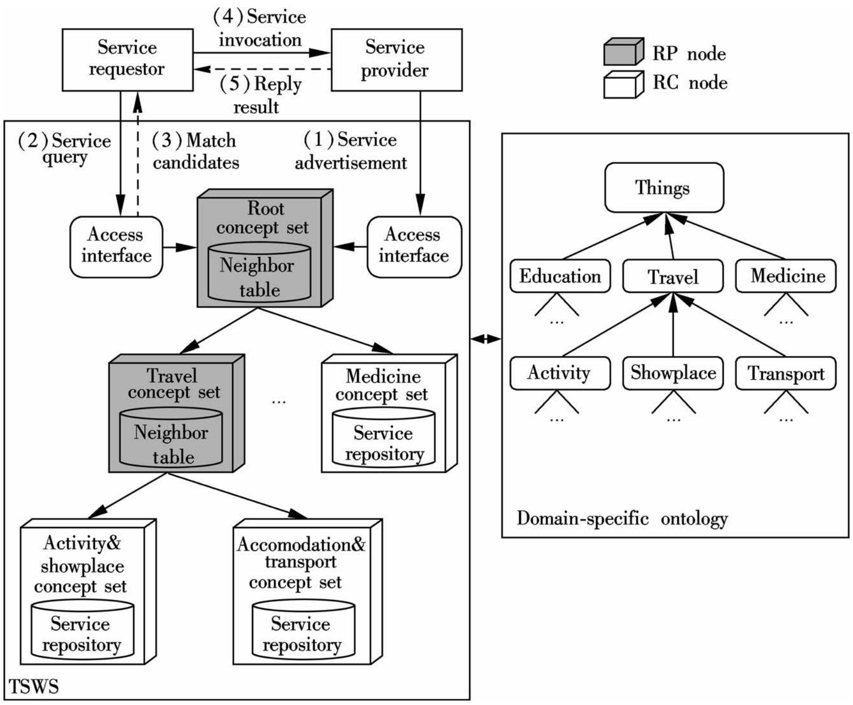

Fig. 1 Topology of the system

2) Registry center (RC) RC nodes are the storage centers in the architecture. Each RC is related to a concept set and the main tasks of the RC are to store the semantic web service descriptions which are related to their concept sets, and to provide appropriate service descriptions in their repositories for user requests.

Each RP node in the system forwards the request message using the neighbor table (see Tab. 1). Also, the balance keeping of the tree is based on the table.

The concept set displays the combination of the children's node concept sets. The depth and layer column can be respectively obtained after the following steps (see Fig. 2, where L represents the layer of the left child, R represents the layer of the left child, S represents the layer of self):

1) The layer of the root node is defined as 0, and the depth of each RP node can be calculated through a top-down traversing process.

2) All the RP nodes respectively copy their own depth to their RC children within the neighbor table. Then when all the layer information of RC nodes is determined, each RP node informs its father the layer information generated from the minimal layer number of children through a bottom-up process.

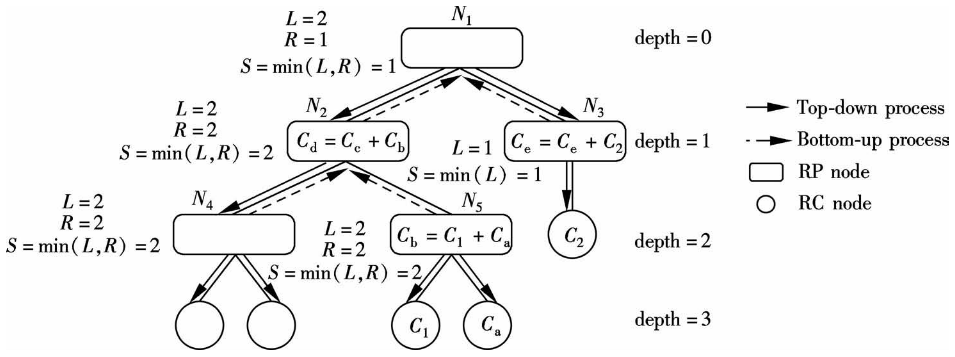

Fig. 2 Layer information achievement

Tab. 1 Neighbor table of N_2 in Fig. 2

Node	Concept set	Address	Layer	Depth
N_1	$\{c_d+c_e\}$	10.3.16.140	1	0
N_2	$\{c_d\}$	10.3.16.141	2	1
N_4	$\{c_c\}$	10.3.16.142	2	2
N_5	$\{c_b\}$	10.3.16.143	2	2

Since there is no registered service in the system, when the first registry request comes, the root will store a new service in the empty RC node as its child. As the number of services goes up, the RP node may utilize the adaptive mechanism proposed in Ref. [12] to dynamically manage its children (RP or RC). When a new RC node is added to the system, the layer information of its ancestors should also be changed through a bottom-up process. In this process, first, each RP node sets the layer information of the child RC node (if there is one) equal to its own depth. Then, after all the children's layer information is generated, the parent node chooses the minimal layer number of its children as its own layer number and reports to its parent. Thus, all the nodes in this system can obtain their own layer information.

The topology is kept as a balance tree based on the adaptive mechanism, since the depth of the tree will not be changed after the joining process if the RP topology tree is not a complete tree. Meanwhile, the leaving process will not lead to imbalance because all the RC nodes in the topology tree are leaves. The amount of messages transmitted during the joining process is limited to $2\log_n N$, where n denotes the maximal number of children of a node, and N denotes the total number of RC nodes in the topology.

The detailed description about topology maintenance is omitted here due to the space limitation, and can be found in Ref. [12].

2.2 Failure tolerant mechanism

Not only registry request but service search request messages are forwarded through the root node of TSWS, and the RP node with lower depth may endure the high probability of forwarding more request messages. We will provide a failure tolerant mechanism to solve the failure problems in TSWS. We suppose that the two continuous nodes in the request path will not fail at the same time; in other words, the probability of that happening is extremely low. The whole architecture is displayed in Fig. 3. In our mechanism, each RP node records all the nodes of his children in case one of his children fails. Once the father node finds the failure of one of its children which the request is supposed to be forwarded to, it will send the request to all of its children in its backup table. We think that the virtual root node in the network should delegate a group of real RP nodes, in which the nodes know each other clearly. In this situation these RP nodes can use several simple classic load-balancing methods such as RRS

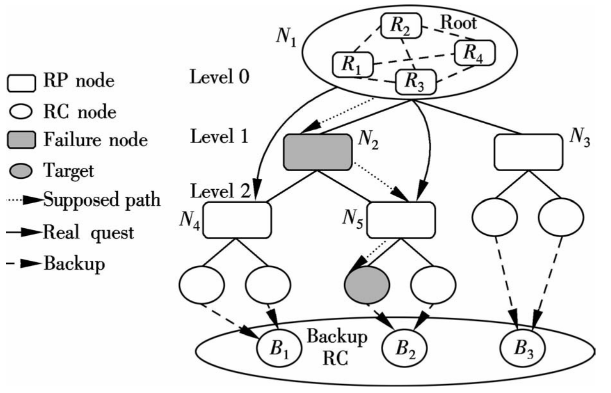

Fig. 3 Failure tolerant mechanism

(round-robin scheduling) or WRRS (weighted round-robin scheduling) to insure the fluency of key nodes, as well as the scalability of our system.

We use the backup RC node to avoid the failure happening in all RC nodes. Supposing that one backup RC node can store the service description of σ RC nodes, the father(s) of those RC nodes should also take note of the existence of the backup one in case the failure occurs in its child node. Also, the RP nodes with the backup tables which contain the RC node should bear that in mind.

3 Semantic Service Matching Algorithm

Receiving a discovery request, the RP node can forward the request to the eligible child node based on the concept set information in the neighbor table. At the final RC node, we employ semantic matching algorithms to look up the probable related web service.

The domain ontology can be described by a graph in which concepts and relationships between them are denoted by graph nodes and edges. Thus similarity between concepts can be calculated through the distance between concept nodes in the graph. An example of an ontology graph is shown in Fig. 4.

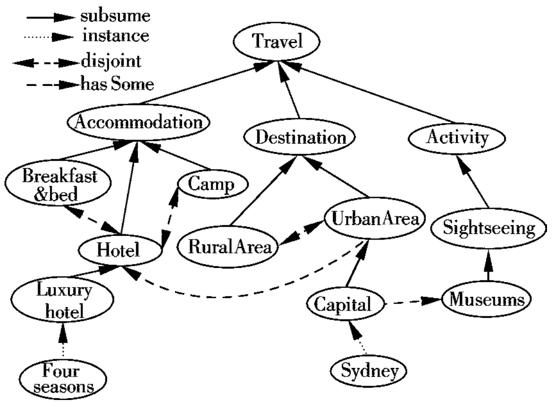

Fig. 4 Example of travel ontology

Different kinds of relationships in the ontology represent different degrees of relevancy between concepts. And distinguishing weights are taken into account when calculating the distance between concept nodes. As shown in Fig. 4, four types of lines represent the relevancy between concept nodes in different sub domains.

However, one special kind of binary relationship "disjoint" has never been considered in existing semantic matching algorithms. Without representing high relevancy between concepts, on the contrary, disjoint relationship only denotes the irrelevancy of concepts. The disjoint edge is always between brother nodes in the ontology graph such as an edge between RuralArea and UrbanArea and an edge between Hotel and Camp in Fig. 4.

It is easy to understand that although RuralArea and UrbanArea are children of the concept destination, the two concepts are totally different. When user request is about UrbanArea, such as a big city, it is unreasonable to provide services related to RuralArea.

Therefore, we weigh different kinds of relationships based on the following rules:

1) Subsume relationship has the lowest weight;

2) Instance relationship has the same weight with subsume relationship;

3) Normal binary relationship between concept nodes with different father nodes has higher weight than subsume relationship;

4) The weight of disjoint relationship is infinite.

Furthermore, the structure information of the ontology should also be taken into consideration. The same kind of relationship at different levels in the ontology can represent diverse degrees of relevancy. In the ontology graph, concepts with higher depth (when only subsume relationship edges are taken into consideration) are more concrete. We assign the weight value of subsume relationships to the edge between concepts according to the following equation[13]:

$$W(a, b) = 1 + \frac{1}{k^{\mathrm{depth}(b)}} \tag{1}$$

where depth(b) represents the depth of concept b from the root concept to node b in the ontology hierarchy; k is a predefined factor greater than 1 indicating the rate at which the weight decreases along the ontology hierarchy.

Altogether, the weight of entire edges in the ontology graph can be calculated as

$$W(a, b) = \begin{cases} 1 + 1/2^{\mathrm{depth}(b)} & P(a, b) = \text{“subsume”} \\ 1 & P(a, b) = \text{“instance”} \\ 2 & P(a, b) = \text{“hasSome”} \\ \infty & P(a, b) = \text{“disjoint”} \end{cases} \tag{2}$$

Fig. 5 shows the result after calculation.

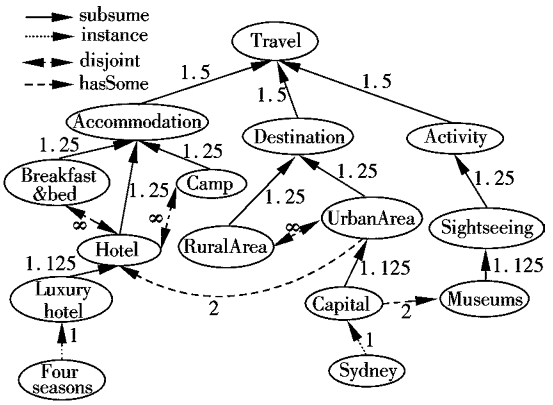

Fig. 5 Weighted ontology graph

After the weight has been obtained, we can calculate the distance between the start node and other nodes through the Dijkstra algorithm. Then the semantic similarity can be measured from the distance as

$$\mathrm{sim}(c_1, c_2) = \frac{t}{\mathrm{dis}(c_1, c_2) + t} \tag{3}$$

where t is a predefined factor no less than 1, which determines the impact degree of semantic distance to semantic similarity with dis(c_1, c_2). We set $t = 1$ empirically. Ultimately, a similarity degree matrix between concepts is generated by SSM, which further supports finding out potential related requested concepts conveniently. Then based on that concept set, the RP nodes can launch the real search of service descriptions.

4 Experimental Evaluation

Simulation experiments are designed to evaluate the performance of the proposed architecture. Before analyzing the experimental results, two important parameters need to be determined. First, one is the child number of each RP node. We suppose that the routing time in each RP node increases logarithmically with the rise of tuples in a neighbor table, and all the RP nodes achieve the status of their children in one unit time. The cost of status maintenance is denoted as

$$c = \frac{\frac{S-1}{n-1} n\alpha}{\frac{S-1}{n-1} n\alpha + m\log_n S(1 + \log_2 n)\beta} \tag{4}$$

where n, S, m, α, β represent the number of children, the number of all the RC nodes, the request processed in unit time, the base request forward process time, and the base status information process time, respectively. Using the geometric progression formula, the number of RP nodes can be obtained from $(S-1)/(n-1)$, and each request process time in the whole procedure is $\log_n S(1 + \log_2 n)\beta$, where $\log_n S$ represents the hops in the query path. We choose the n to make the lowest cost in unit time, and as a result, n is set to be 13.

Another major parameter is the capacity of each RC node. As we can imagine, the process time within each RC node will definitely grow up with the increase in the requests to the node. We use ten thousand RC nodes to measure the growth property of the average response time under different capacities (see Fig. 6).

As we can see, the response time has an exponential relationship with the capacity of the RC node. The response time exceeds 1 s when the capacity is over 600, and it will exceed the acceptable response time when the capacity becomes larger. We take 500 as the applicable value in our next experimental environment.

4.1 Routing performance and scalability evaluation

To evaluate the routing performance, the average hops required for a discovery query according to the scale of our architecture are measured. We limit each RC node to contain one single concept with minimum granularity. So

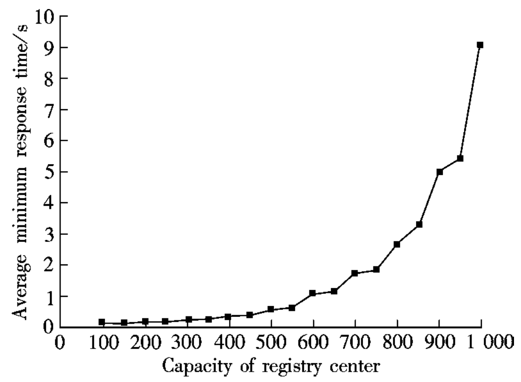

Fig. 6 Average minimum response time

in the Chord4S system, each RC node can map to a single peer. Also, we choose four times more peers without any service descriptions in Chord4S to simulate the P2P environment. It can be seen from Fig. 7 that our TSWS has a lower hops cost in the procedure of service discovery. In addition, the scalability can be recognized from Fig. 7. With the rapid growth of the RC node, the average hops only slightly increase due to the tree topology.

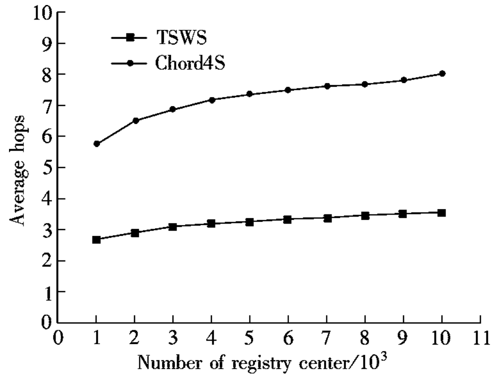

Fig. 7 Routing performance

Moreover, the failure tolerant mechanism significantly decreases the probability of failure lookups in our TSWS. We use the same experimental environment as before. As shown in Fig. 8, both methods hold very low failure lookups when small proportional nodes fail. With the addition of the failed node, the Chord4S method has more failed lookups than our proposed method. The most meaningful property of our TSWS is that it maintains a very low failure rate (less than about 0.1) when the failure node rate is less than 20%. In reality, this property will help our topology run more robustly.

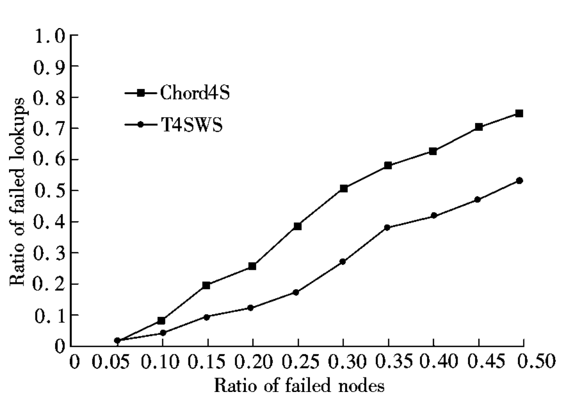

Fig. 8 Failure tolerant mechanism performance

4.2 Semantic matching evaluation

In order to detect how our SSM performs, we compare the recall and precision results with those obtained by the grade matching algorithm and the algorithm proposed by Fu et al. The test set is services in travel domain from OWL-S service retrieval test collection version 2(OWLS-TC v2) [10].

As shown in Fig. 9, our algorithm has both better recall and precision than the other two algorithms. For the same precision, the recall of our result is about 15% higher than that of the grade matching algorithm and about 8% higher than that of Fu's algorithm. Under the same recall, the precision of our result also exceeds the other two algorithms.

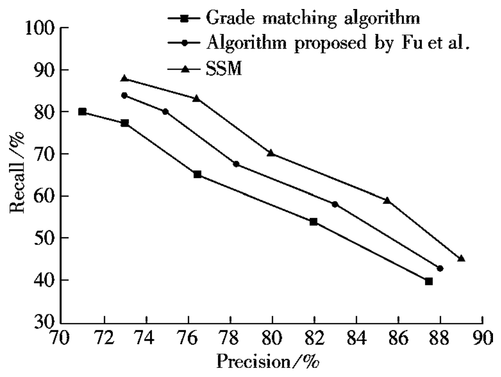

Fig. 9 Comparison of service matching algorithm

5 Conclusion and Future Work

In this paper, we present a distributed system named TSWS to address the problem of scalable service registry and discovery. The scalability of our method is ensured by means of maintaining a balance-tree like topology, which provides more efficient service registry or discovery for users. Meanwhile, a failure tolerant mechanism is equipped to avoid damage caused by single node failure. We take advantages of common ontology to derive the concept set from request, which is further used to locate the target RC node. Results shown in Fig. 9 experimentally prove that the proposed method has better performance in recall and precision than the other two competitive algorithms. The introduced relationships of "disjoint" and "hasSome" not only promote the efficiency but also the accuracy.

References

[1] Sycara K, Paolucci M, Ankolekar A, et al. Automated discovery, interaction and composition of semantic web services[J]. *Journal of Web Semantics*, 2003, **1**(1): 27

−46.

[2] Christensen E, Curbera F, Meredith G, et al. Web services description language (WSDL) 1. 1 [EB/OL]. (2001-03-15)[2011-08-30]. http://www. W3. org. /TR/wsdl.

[3] Clement L, Hately A, Riegen C, et al. UDDI version 3.0.2 [EB/OL]. (2004-10-19) [2011-10-20]. http://www. uddi. org/pubs/uddi_v3. htm.

[4] Srinivasan N, Paolucci M, Sycara K. An efficient algorithm for OWL-S based semantic search in UDDI [C]// *International Workshop on Semantic Web Services and Web Process Composition*. San Diego, USA, 2004: 96 − 110.

[5] Nan B, Li S, Yang S. The study of distributed semantic web services' publishing and discovery mechanism [C]// *International Conference on Communications and Intelligence Information Security*. Ganzhou, China, 2010: 51 − 54.

[6] Burstein M, Bussler C, Finin T, et al. A semantic web services architecture [J]. *Internet Computing*, 2005, **9**(5): 72 − 81.

[7] Martin D, Burstein M, McDermott D, et al. Bringing semantics to web services with OWL-S [J]. *Journal of World Wide Web*, 2007, **10**(3): 243 − 277.

[8] Paolucci M, Kawmura T, Payne T, et al. Semantic matching of web services capabilities [C]//*The International Semantic Web Conference*. Seattle, USA, 2002: 333 − 347.

[9] Fu Pengbin, Liu Shanshan, Yang Huirong, et al. Matching algorithm of web services based on semantic distance [C]// *International Workshop on Information Security and Application*. Qingdao, China, 2009: 465 − 468.

[10] Klusch M, Fries B, Sycara K. Automated semantic web service discovery with OWLS-MX [C]//*Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*. Hakodate, Japan, 2006: 915 − 922.

[11] He Qiang, Yan Jun, Yang Yun, et al. Chord4S: a P2P-based decentralised service discovery approach [C]// *IEEE International Conference on Services Computing*. Honolulu, HI, USA, 2008: 221 − 229.

[12] Yao Y, Cao J X. Scalable mechanism for semantic web service registry and discovery [J]. *Journal of Southeast University*: *Natural Science Edition*, 2010, **40**(2): 264 − 269. (in Chinese)

[13] Ge J, Qiu Y. Concept similarity matching based on semantic distance [C]//*International Conference on Semantics, Knowledge and Grid*. Beijing, China, 2008: 380 − 383.

基于公共本体的高效语义服务发现

曹玖新[1,2] 秦 屹[1,2] 张 松[3] 刘 波[1,2] 东 方[1,2]

([1] 东南大学计算机科学与工程学院, 南京 211189)
([2] 东南大学网络和信息集成教育部重点实验室, 南京 211189)
([3] 南京师范大学, 南京 210042)

摘要:为解决集中式服务发现结构存在的性能瓶颈问题,基于领域本体语义信息,提出一种能自适应地调整领域划分、分配系统资源的分布式 web 服务发现体系结构,并分析了该结构的可扩展性、自组织性和自适应性. 具体描述了该结构下的语义 web 服务发现算法的 2 个阶段:语义注册中心定位和基于输入输出的服务匹配. 在注册中心组成的平衡树拓扑结构中,注册代理能够快速将请求转发至目标注册中心,避免产生性能瓶颈. 然后,通过引入一种新的基于语义距离的服务匹配算法来进行服务查询效果优化. 模拟实验结果表明:提出的服务发现方法具有高可扩展性的优点;与其他服务查询算法相比, 服务匹配算法具有更高的查全率和查准率.

关键词:服务发现;领域本体;语义 web 服务; 语义距离
中图分类号:TP393.0