A Scalable and Adaptive Distributed Service Discovery Mechanism in SOC Environments

Xiao Zheng, Junzhou Luo, and Aibo Song

School of Computer Science and Engineering, Southeast University,
210096 Nanjing, P.R. China
{xzheng,jluo,absong}@seu.edu.cn

Abstract. Current researches on service discovery mainly pursue fast response and high recall, but little work focuses on scalability and adaptability of large-scale distributed service registries in SOC. This paper proposes a solution using an agent based distributed service discovery mechanism. Firstly an unstructured P2P based registry system is proposed in which each peer is an autonomous registry center and services are organized and managed according to domain ontology within these registry centers. Secondly, an ant-like multi-agent service discovery method is proposed. Search agents and guide agents cooperate to discover services. Search agents simulate the behaviors of ants to travel the network and discover services. Guide agents are responsible to manage a service routing table consisting of pheromone and hop count, instructing search agents' routing. Experimental results show that the suggested mechanism is scalable and adaptive in a large-scale dynamic SOC environment.

Keywords: multi-agent system, P2P network, service discovery, ant algorithm.

1 Introduction

In Service-Oriented Computing (SOC), in order to discover and locate a target service efficiently, services should be published to a service registry. The registry stores metadata documents of Web services, including functions, parameters and providers. The service registry is a bridge between service consumers and service providers.

Currently used services registries, for example those based on UDDI standard, often adopt a centralized or hierarchical architecture, which are not suitable for very large SOC environments for their intrinsic poor scalability features. The design of decentralized registry systems is therefore urgent. Recent UDDI v3 [1] introduces a mechanism of registry affiliation. Affiliated registries could share data with each other in a controlled environment. Peer-to-peer(P2P) based registry is also suggested[2-6] to support distributed service discovery. The P2P registry architectural style has no centralized registry to store the metadata of services. Each peer in the P2P registry is a registry center that maintains data independently, and data can be shared among peers. Recent researches almost focus on data partition and management among peers, and designing a high efficient service discovery algorithm. Little work considers scalability and adaptability of the registry system when thousands of

J. Cao et al. (Eds.): NPC 2008, LNCS 5245, pp. 349–360, 2008.

peers exist in it. However, the problem of scalability and adaptability must be solved for distributed computing.

Under an environment consisting of thousands of registries, service discovery would involve locating the correct registry in the first place and then locating the appropriate service within that registry. This paper focuses on solving the first challenge of finding appropriate registries. We propose an adaptive distributed services discovery mechanism for large-scale dynamic SOC environments. The main contributions in this paper are:

(1) We propose an agent based distributed service registry system, which is based on unstructured P2P architecture. Moreover, domain ontology is adopted to partition and manage services in registry centers.

(2) We examine a decentralized and self-organizing approach inspired by ant behaviors. According to dynamic variable pheromone level, agents can adapt to the changes of registry topology and registered services.

2 Related Work

Distributed service publication and discovery models have been extensively studied in previous work. Many researches [2-6] suggest using P2P network as the infrastructure of service registries. Current P2P systems can be classified into two types, namely unstructured and structured. Structured designs are likely to be less resilient in the face of a very transient user population, precisely because the structure required for routing is hard to maintain when nodes joining and leaving at a high rate. In contrast, the unstructured networks are ad-hoc and the placement of data is completely unrelated to the overlay topology. The advantage of such systems is that they can easily accommodate a highly transient node population. In addition, unstructured networks support many desirable properties such as simplicity, robustness, low requirement for network topology and supporting semantic searching.

METEOR-S [2] is a scalable P2P infrastructure of registries for semantic publication and discovery of Web services. This work uses an ontology-based approach to organize registries into domains, enabling domain based classification of all Web services. The mapping between UDDI registry and ontology is used to organize a P2P network. Under this mechanism, the content of the registry is tightly coupled with the topology of P2P network, which leads to their synchronization and less flexibility. Reference [3] suggests a P2P and semantic web based service discovery mechanism where deployment and publication of a Web service are bound together. This mechanism omits obvious publication process of services, and could get dynamic various QoS of services in time. However, due to lacking registry, the response time of service discovery certainly delays, and its efficiency must lower. Reference [4] proposes a P2P based service discovery mechanism by creating an agent called P2P Registry as a middleware within DNS and peers. Each peer is able to register and discover desirable services automatically through current DNS within a short duration. In this scheme, centralized DNS is used to locate the target, which is similar to the super node in hybrid P2P systems.

3 A P2P-Based Distributed Service Registry System

In order to support dynamic adaptive service discovery, an agent based unstructured P2P registry system is introduced, which is illustrated in figure 1. The whole registry system is composed of numerous self-organized registry centers that are interconnected through unstructured P2P network style. Three types of agents are used to manage registry centers and discovery services, namely registry agent, search agent and guide agent (RA, SA, GA for short respectively). RA, resided in registry centers, is responsible for registering and indexing service metadata, etc.. SA is a kind of mobile agent, accepting discovery task and responsible for searching target services over the registry system. GA, also resided in a registry center, is a guide who helps SA to select a best route. GA maintains a service routing table which records routes from the local registry to service domains.

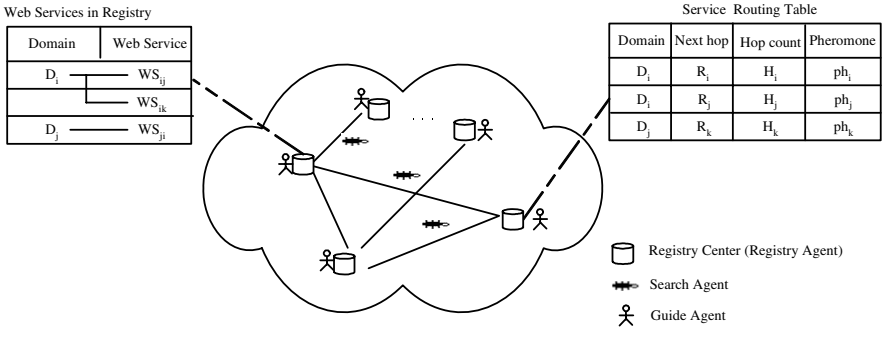


Fig. 1. An overview of the distributed service registry system

Widely adoptive UDDI standard only supports keyword-based search. However, the accepted view is that semantic organization and management should be supported in near future, which should guarantee the accuracy of search result and offer foundation to automatic service composition based on semantics. Our suggested registry center adopts a domain ontology based two-level structure.

Domain ontology could support the common understanding about domain knowledge, and eliminate different meanings of the same word or sentence [8]. In SOC, a service always belongs to one or more particular domains. For example, car rental services belong to a traffic and transport domain, and hotel booking services belong to a travel domain. Consequently, registered services can be categorized in terms of which domain them belong to. Each registry center manages and clusters services by means of domain ontology.

A service can be defined as $WS = (I, O)$. I is a set of inputs, and O is a set of outputs. For $WS_i(I_i,O_i)$, if I_i and O_i belong to domain D_i semantically, WS_i will be registered in D_i. Once the registry center does not contain such D_i, a new domain D_i will be added. As illustrated in figure 1, registered services are organized as a directory tree. When querying a service WS_{ij}, a conclusion can be drawn rapidly through judging whether it belongs to one domain of the registry center or not. Consequently, the

query process can be divided into two stages. At the first stage, which domain the target service belonging to should be judged; at the second stage, the service could be queried within its domain. Index technique can be used to improve the speed of query. Semantic technique can be used to judge similarity between a service and a domain. All of these techniques can be referred in the study of information retrieval and semantic web, which will not be studied in this paper.

4 Agent-Based Ant-Like Service Discovery

Ant algorithm, based on behaviors of the real ant, is initially applied to find the shortest paths in a graph [7] and, later on, successfully applied to combinatorial problems [9] or network routing [10]. The principle of this class of artificial ant algorithm is to translate into algorithmic models some of the real ants' biological principles. They use pheromone remaining in paths to indirectly exchange the path information between ants, whereas former ants passing by the path, which represents some experience knowledge, deposit such pheromone. In ant algorithm, the positive feedback of global updating reduces the search scope, and the hidden negative feedback retains the scope.

Our work also follows these thoughts. For each query request, n SAs are generated which emulate ants to execute a query task. In distributed service registry architecture, different service providers usually register many services with the same functions but different QoS properties. Service consumers do not generally find one particular service, but a number of services belonging to a given service class, so that they can subsequently select the service which is the best suited for their applications. A service class can be seen as a set of services satisfying a given set of syntactic and semantic constraints on the values of service metadata parameters. The objective of SAs is therefore to find target services as many as possible under particular constraints.

4.1 Behaviors of Agents

SA roams over the P2P network and queries the services belonging to its own ontology domain. Each SA carries some property information including a target service, TTL(Time to live), hop count and Tabu (tabu table). TTL records the life-span of SA, and hop count is the number of nodes (registry centers) that a SA has passed after discovering the latest target service. Tabu contains all nodes having been visited. SA's behaviors are described as follows. Corresponding algorithms will be introduced in section 4.2.

(1) Roam. In order to find target services, SA moves among nodes by a predefined routing policy. Each node maintains a service routing table that directs SA to select next hop. As illustrated in figure 1, a service routing table includes four fields, namely target domain, next hop, hop count and pheromone. The first record in service routing table showed in figure 1 represents that if target services belong to domain D_i, R_i will be the next hop where the distance from local to the node having D_i is hop count H_i, and the amount of pheromone on this exit is ph_i. In order to avoid visiting the same node again, SA records the visited nodes in its Tabu. At last the hop count is updated.

(2) Querying services. SA queries target services at the visited node. The two stage query approach has been simply introduced above.

(3) Generating and sending pheromone updating messages. A message includes service domain and hop count. There are two different kinds of updating messages. If SAs having found successfully, the massage of reporting a target service registered in this node is going to be flooded to all its neighbors. In addition, if SA's hop count is not equal to zero, a message containing this hop count will be sent to the local GA.

(4) Life-span control. SA has a life-span, which could control the number of SAs roaming in the network and insure there are no more SAs moving ceaselessly after a query is over. When a SA is created, its TTL is set to an initial life value. After this, TTL will be updated by a particular rule. The number of SAs roaming over the network can therefore be controlled. In addition, by adjusting the initial life value, SA's searching radius can be increased or decreased. Because SA can destroy itself after its TTL is decreased to zero, its life-span could be managed by itself.

GA mainly maintains the service routing table. Its behaviors are described as follows:

(1) Listening and receiving updating messages. GA always listens in pheromone updating messages from SAs or neighbor nodes' GAs.

(2) Managing service routing table. Updating messages are accepted and analyzed. If the service domain specified in the message could be found in the routing table, pheromone and hop count in the corresponding item will be updated. Conversely, if the domain does not exist, a new item will be added. Hop count contained in the accepted updating message will replace the hop count in the current item. Due to the dynamic variety of service availability, pheromone is always decreased periodically. The pheromone, which does not increase after a long time, would be given out at last. It denotes that no service exists in the corresponding routing, or no requirements about searching this kind of services exist. If an item's pheromone is zero, it will be deleted from the table.

(3) Diffusion of pheromone updating message. When resided node joins a network, or a new service is registered successfully, GA floods messages to all its neighbors with hop count equal to 1.

4.2 Ant-Like Service Discovery Approach

4.2.1 Search Agent Routing Policy

When SA queries the current service routing table, two cases would appear. The first case is that target services do not belong to any service domains in the table. In such situation, SA would select a neighbor randomly. The second case is that one or more domains can match the target services. Here SA would make a decision in terms of the amount of pheromone and hop count in corresponding items. The amount of pheromone denotes the number of ever-successful search along the route, and hop count denotes the distance to target services. SA moving to the neighbor with more pheromone may get higher success rate, and selecting less hop count may get shorter response time. Therefore, two factors should be considered together. Because pheromone represents rather a probability than certain knowledge, a roulette wheel selection algorithm [11] is used to select a neighbor. After adopting this method, the path with fewer pheromones also has the chance to be chosen, even if the probability is smaller.

In the second case, the probability of SA k at node i choosing to move to neighbor j is defined as

$$p_k(i,j) = \begin{cases} \dfrac{ph(i,j)(1/hops(i,j))^{\lambda}}{\sum_{u \in N} ph(i,u)(1/hops(i,u))^{\lambda}}, & j \in N \\ 0, & j \notin N \end{cases} \quad (1)$$

$$let \quad N = \{t[Nexthop] | t \in RT_i \wedge s \in t[Domain]\} - Tabu(k)$$

where $ph(i,j)$ denotes the amount of pheromone from node i to node j, $hops(i,j)$ denotes the hop count which is the distance to target service domain while selecting j as the next hop, and $\lambda > 0$ is a parameter which determines the relative importance of pheromone versus distance. s denotes the target service, RT_i denotes the service routing table in node i, $t[Domain]$ denotes the projection on field $Domain$, $t[Nexthop]$ denotes the projection on field $Nexthop$. $Tabu(k)$ is the Tabu of SA k.

4.2.2 Pheromone Generation and Updating

Pheromone, which directs SA to select routing, plays an important role in SA routing. Thus, how to generate and update pheromone is a key to influence algorithm performance. In the following cases, pheromone will be generated or updated.

(1) SA will deposit pheromone on the path passed by if it has discovered a target service. Its generated new pheromone will be added to the pheromone remained on the path previously. Assume that SA enters local node i from neighbor n, and let $ph(i,n)$ denotes the pheromone amount on the path from local node i to neighbor n, a updating rule is give by formula (2)

$$ph(i,n) = \alpha \cdot ph(i,n) + (1-\alpha)\Delta p_1 \quad (2)$$

where $\alpha \in (0,1)$ and Δp_1 is a constant.

(2) When having found a target service at node n, SA would diffusion pheromone to all neighbors. Receiving a message of pheromone updating, GA positioned on its neighbor would update pheromone in the corresponding item of its service routing table by formula (3).

$$ph(m,n) = \beta \cdot ph(m,n) + (1-\beta)\Delta p_2, \quad m \in J(n) \quad (3)$$

where $\beta \in (0,1)$ and Δp_2 is a constant. m is a element of the set of node n's neighbors. $J(n)$ is the set of node n's neighbors.

(3) For each item of the service routing table, an update process will be done in period by formula (4).

$$ph(i,n) = \rho \cdot ph(i,n) \quad (4)$$

where $\rho \in (0,1)$ is pheromone decay parameter. Thus if a neighbor is always not be visited, its pheromone level will be closer to zero.

(4) If a new node enters the P2P network or new services have been registered in a node, the local GA will send update messages to all its neighbors. In this situation, neighbors' service routing tables will be updated by formula (3).

4.2.3 SA's Hop Count Updating

When SA searches a target service, it records the hop count from latest discovered service to current position. The initial value of SA's hop count is zero, which represents it has not discovered any target services. When SA finds a target for the first time, it will set hop count to 1. Hereafter, once discovering a target service at a node, SA will reset hop count to 1; otherwise add 1 to current hop count. Such hop count is contained in pheromone updating messages and is used to update service routing table. Hop count HC_k is updated by (5)

$$HC_k = \begin{cases} 0, & HC_k = 0 \wedge s_k \notin C(i) \\ 1, & s_k \in C(i) \\ HC_k + 1, & HC_k > 0 \wedge s_k \notin C(i) \end{cases} \tag{5}$$

where s_k denotes the target service of SA k, and $C(i)$ denotes the service set of node i.

4.2.4 Life-Span Control

Our algorithm uses TTL to control SA's life-span. When a SA is created, its TTL will be set to an initial value. Hereafter at each visiting node, the SA's TTL will be updated. If SA does not find target services, its TTL will be decreased; otherwise it will not be changed so that the SA can visit more nodes. If all neighbors are in the Tabu, the SA's TTL will be set to zero. Such SA will not move any more and be killed. When SA k at node i, the update rule of its TTL_k is:

$$TTL_k = \begin{cases} TTL_k - 1, & s_k \notin C(i) \\ TTL_k, & s_k \in C(i) \\ 0, & J(i) \subseteq Tabu(k) \end{cases} \tag{6}$$

where the meanings of s_k and $C(i)$ are similar to formula (5).

Based on discussion above, the routing selection algorithm is described as follows.

Algorithm 1. Routing selection algorithm. SA runs this algorithm at each node.

```
1:  Input Seach Agent SA, target servies WS, local node
lnode, pheromone increment ph1 and ph2;
2:  SendUpdatePheromoneMessage to lnode with
SA.HopCount and ph1;
3:  SA.UpdateHopCount;
4:  SA.Tabu.Add(lnode);
5:  IF Other SAs having the same task had arrived THEN
6:      SA.TTL--;
7:  ELSE
8:      SA.Query(WS);
9:      IF (NoFound) THEN
10:        SA.TTL--;
11:     ELSE
12:        FOREACH node_i IN lnode.neighbour
13:          SendUpdatePheromoneMessage to node_i with
msg.hc=1 and ph2;
14:     END IF
```

```
15:  END IF
16:  IF All lnode.neighbour in SA.Tabu THEN
17:     SA.TTL=0;
18:  ELSE
19:     SA.Nexthop =RouletteSelect(lnode);
20:  END IF
```

5 Simulation and Performance Analysis

This section evaluates performance of our approach. For the sake of focusing on SA's search performance over the P2P based registry, assume that once SA reaches a node owning target services, it could discover all of them.

Repast 3(REursive Porus Agent Simulation Toolkit) [12] is a multi-agent simulation platform for large-scale systems. Our simulation program is based on Repast and implemented in C#. Simulated network topology is a Power-law random graph[13]. Each registry center has at most 20 kinds of service domains, and each domain has at most 50 kinds of services. 200 services are randomly generated and registered in each registry center. The class of ant-based algorithms often uses a number of tuning parameters. Unfortunately, these parameters are not directed by scientific rigor theory until now [14]. According to repeating experimentation and guidelines in reference [15], the parameters are set as α=0.9,β=0.85,ρ=0.95,Δp_1=0.25, and Δp_2=0.35 in our algorithm. Parameter λ determines whether pheromone or hop count more affects routing selection. In a static environment, where network topology and metadata of registered services are generally stable, hop count could truly reflect the position of registered services. In contrast, the mechanism of updating pheromone plays an important role for adapting the algorithm to dynamic environments. Therefore, in static simulations, λ is set to 0.5, in order to magnify the effect of hop count, but λ=2 in dynamic simulations. The simulation is based on a discrete time model and all events are executed in discrete time steps, called tick.

Two groups of experiments are designed to evaluate our algorithm roundly. The first group examines the performance of the algorithm, and how the number of SAs and TTLs affect its performance. The second compares our algorithm to two classical resource discovery algorithms in P2P systems.

We focus on scalability and adaptability in a large-scale dynamic environment, and use the following metrics:

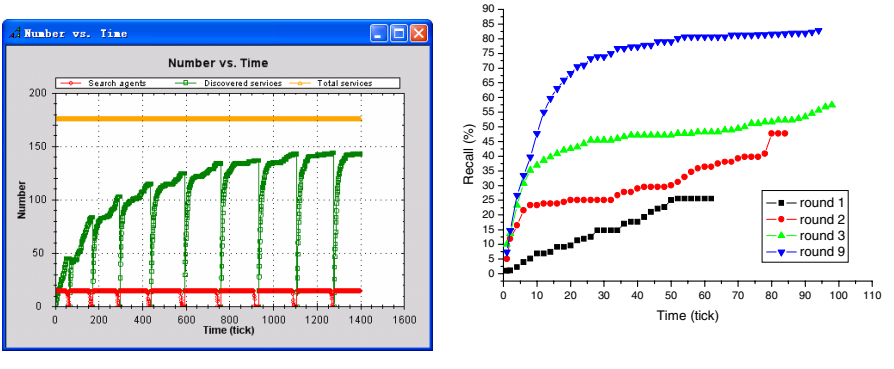
- *Recall* is the ratio of the number of discovered target services to the total number of target services, which shows the algorithm's search capability.
- *Search performance-price ratio* is average number of target services discovered by a search agent or query message. It is calculated as (number of discovered target services)/(number of query messages or SAs).

5.1 Performance Analysis

First experiments run in a static system in which the number of registry center, network topology, and the kind and number of services are invariable. Figure 2 shows a scenario, in which there lacks pheromone initially, and SAs repeatedly search a same

target service later. There are 175 target services randomly distributed in the system of 1000 registry centers. 15 SAs with initial TTL=20 are generated to search target services. A search round is the span from the generation to death of all SAs for the same target. Figure 2(a) is a real-time statistic graph, which shows the trend of the number of SAs and discovered services along with running time. At the first search round, recall is very low and reaches 26%. However from the second round, a higher recall can be reached in few time steps. At the tenth round, a recall of about 83% can be reached. This is because at the beginning there is no pheromone in the service routing table, so SA blindly selects routing. After several rounds, pheromone has accumulated which could direct SA to search the registry with higher probability of owning target services. Summing up the data in figure 2(a), figure 2(b) shows the comparison between recalls of different rounds. Because Figure 2 illustrates that a fast convergence and a high recall can be reach in a short time in our algorithm.

Figure 3 shows how different system size affects the recall. As the number of registry centers increasing, recall decreases slowly. Because the number and initial TTL of SAs do not change, the degressive trend is rational. Figure 4 shows how the number and initial TTL of SAs affect the recall when the number of registry centers is 4000. Increasing the number of SAs can improve search scope, and increasing initial TTL can increase search radius. One of approaches of improving recall is therefore to increase the number and initial TTL of SAs, especially when the system size increases. Figure 2, 3, and 4 show that our algorithm is feasible for large-scale service discovery systems.



(a) (b)

Fig. 2. Results of repeated running. (a) Trend of the number of discovered services and SAs. (b) Comparison between recalls of different rounds.

In order to examine the performance of our algorithm in dynamic environments, the experiment simulates a dynamic system where the number of registry centers is varying. For the sake of getting stable statistic, assume that there are no changes during a search round and randomly increase or decrease 10% registry centers between search rounds. Figure 5 is a dynamic output graph of the simulation program when the number of registry centers is 500. It shows the changing number of discovered services and total services after 10% registry centers having been increased. Figure 5(b)

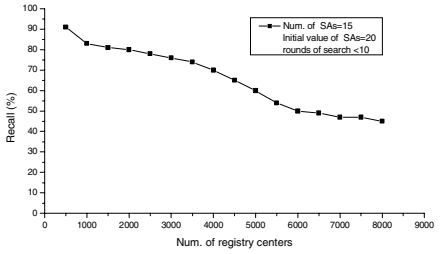


Fig. 3. Recall versus the number of registry centers

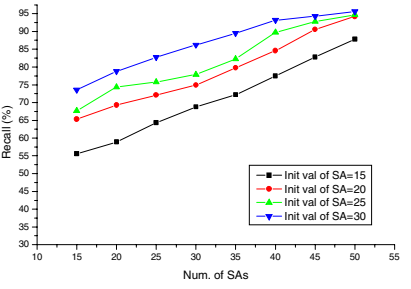
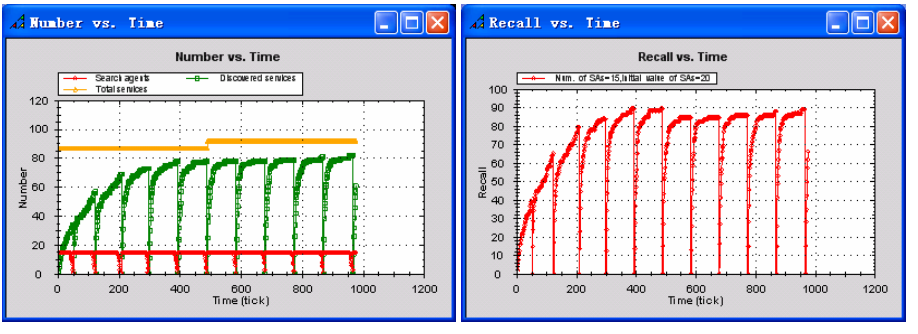


Fig. 4. Recall versus the number of SAs



(a) (b)

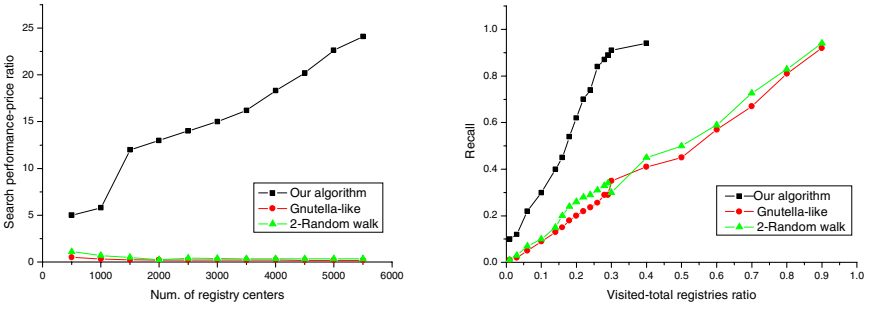
Fig. 5. Experiments in a dynamic environment. (a) Process of multi-round search. (b) Trend of recall.

shows that the recall decreases to 85%, but is back to former 90% after 3 rounds. Obviously, our algorithm is adaptable to the change of the system scale.

5.2 Comparison with Classical Algorithms

Gnutella[16] and Random Walks[17] are classical resource discovery algorithms, which are often used to P2P based service discovery model. Figure 6(a) shows the comparison of search performance-price ratio under different system scales. Since Gnutella's flooding style generate many new SAs at each node, and k-Random Walks generate k-1 new SAs, numerous SAs will be generated in order to get high recall. As a result, their search performance-price ratio is very low. Conversely, as our algorithm does not generate new SAs during the process of searching, a high search performance-price ratio is easily gained.

Again, as showed in Figure 6(b), through the pheromone-based instruction, our algorithm can obtains high recall after visiting about 30% registry centers. However, the other two algorithms' recall is geometric proportion to visited registry centers. This is because our algorithm has the knowledge about which registry probably having target services. The other algorithms only depend on magnifying search scope to improve recall.



(a) (b)

Fig. 6. Comparison between three algorithms. (a)Search performance-price ratio versus the number of registry centers. (b) Recall versus the visited to total ratio of registry centers.

6 Conclusions and Future Work

This paper proposes an agent based distributed service discovery mechanism for a P2P based registry. Under this model, search agents and guide agents cooperate to discover services. Search agent simulates the behaviors of ant to travel the network and discover services. Guide agent is responsible to manage a service routing table. The self-organizing and decentralized nature of the involved algorithms, along with the analysis of performance results obtained with variable system sizes, shows that the proposed mechanism is scalable and adaptive and can be adopted in a large-scale dynamic computing environment.

Recently some two layered and virtual domain based P2P models have been suggested to construct a distributed registry, which try to organize and manage services more efficiently. A service discovery approach under such infrastructure will be studied and implemented in near future.

Acknowledgments. This work is supported by National Natural Science Foundation of China under Grants No. 90604004 and 60773103, Jiangsu Provincial Natural Science Foundation of China under Grants No. BK2007708, Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No. BM2003201 and Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education under Grants No. 93K-9.

References

1. Clement, L., Hately, A., Riegen, C.V., Rogers, T.: Universal Description Discovery & Integration (UDDI) 3.0.2 (2004), `http://uddi.org/pubs/uddi_v3.htm`
2. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J.: METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. Journal of Information Technology and Management 6(1), 17–39 (2005)

3. Chen, D.W., Xu, B.C., Yue, R., Li, J.Z.: A P2P Based Web Service Discovery Mechanism with Bounding Deployment and Publication. Chinese Journal Of Computers 28(4), 615–626 (2005)
4. Chen, C.W., Gan, P.S., Yang, C.H.: A Service Discovery Mechanism with Load Balance Issue in Decentralized Peer-to-peer Network. In: 11th International Conference on Parallel and Distributed System(ICPADS 2005), pp. 592–598. IEEE Press, New York (2005)
5. Liu, Z.Z., Wang, H.M., Zhou, B.: A Two Layered P2P Model for Semantic Service Discovery. Journal of Software 18(8), 1922–1932 (2007)
6. Guo, D.K., Ren, Y., Chen, H.H., Luo, X.S.: A QoS-Guaranteed and Distributed Model forWeb Service Discovery. Journal of software 17(11), 2324–2334 (2006)
7. Dorigo, M., Gambardella, L.M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Trans. on Evolutionary Computation 1, 53–66 (1997)
8. Arpinar, I.B., Zhang, R.Y., Aleman-Meza, B., et al.: Ontology-Driven Web Service Composition Platform. In: Proc. of the Int'l Conf. on E-Commerce Technology, San Diego, pp. 146–152. IEEE Press, New York (2004)
9. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York (1999)
10. Caro, G.D., Dorigo, M.: Antnet: Distributed Stigmergetic Control for Communications Network. Journal of Artificial Intelligence Research 9, 317–365 (1998)
11. Baker, J.E.: Reducing Bias and Inefficiency in the Selection Algorithm. In: The Second International Conference on Genetic Algorithms and their Application, pp. 14–21 (1987)
12. North, M.J., Collier, N.T., Vos, J.R.: Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. ACM Trans. on Modeling and Computer Simulation 1, 1–25 (2006)
13. Adamic, L.A., Lukose, R.M., Puniyani, A.R., Huberman, B.A.: Search in Power Law Networks. Phys. Rev. E64, 46135–46143 (2001)
14. Ridge, E., Curry, E.: A Roadmap of Nature-inspired Systems Research and Development. Multiagent and Grid Systems 3, 3–8 (2007)
15. Parunak, H.V.D., Brueckner, S.A., Matthews, R., Sauter, J.: Pheromone Learning for Self-Organizing Agents. IEEE Trans. on Systems, Man, and Cybernetics-Part A: Systems and Humans 35(3), 316–326 (2005)
16. Gnutella website, `http://www.gnutella.com`
17. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and Replication in Unstructured Peer-to-peer Networks. In: The Int'l Conf. on Measurements and Modeling of Computer Systems, pp. 84–95 (2002)