A Centralized Service Discovery Approach for Agent-based Cloud Computing System

Dingzheng Liu, Weiwei Xing*, Xiaoping Che and Peng Bao

School of Software Engineering, Beijing Jiaotong University, Beijing 100044, P.R. China

Abstract: Agent-Based Cloud computing is emerging as a paradigm in many domains, including service discovery, service composition and service negotiation, etc. It achieves autonomous resource allocation and resource management by delegating requests to agents. Agents utilized the obsolete acquaintance network and Service Capability Tables (SCTs) to record and schedule agents for service discovery. These approaches are unable to perform global search under the intermittent and volatile cloud computing environment. In this paper, we propose a robust service discovery approach with local and global service discovery capabilities. We validate the proposed approach in different granularity of agent network scale. Experimental results demonstrate that our approach outperforms the SCTs based approaches in the aspect of data availability consistently.

Keywords: Agent-based system, cloud computing, service discovery.

1. INTRODUCTION

A cloud is distributed technology platform that leverage sophisticated technology innovations to provide highly scalable and resilient environments, which can be remotely utilized by organizations in a multitude of powerful ways [1]. It is a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms, and/or services). These resources can be dynamically reconfigured to adjust to a variable load, allowing for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model, in which guarantees are offered by the Infrastructure Provider by means of customized Service-Level Agreement (SLAs) [2]. Cloud computing has developed rapidly because of its flexibility to manage resource without any client side hassle. Clients can easily add or remove capacity to form a desirable system for their own specific usage, because of the resource pooling and resource sharing technology adopted by cloud providers.

An agent is a computational entity that acts on behalf of another entity (or entities) to perform a task or achieve a given goal [3]. Multi-Agent system represents a computing paradigm based on multiple interacting agents that are capable to intelligent behavior [4]. Agent-based cloud computing [5] is a multi-agent based paradigm introduced to manage the cloud resource autonomously. Agent-based cloud computing is concerned with the design and development of software agents for bolstering cloud service discovery, service negotiation, and service composition [6]. Based on the works of [7], we illustrate the architecture of agent-based cloud computing system in Fig. (**1**).

In the architecture, different agents are proposed to represent different roles. The system consists of four kinds of agents: Resource Agents (RAs), Service Provider Agents (SPAs), Broker Agents (BAs) and Consumer Agents (CAs).

RAs play the role of managing and controlling access to web services. RAs receive requests from SPAs or other RAs, and carry out these requests by accessing their associated web services. Finally, RAs transmit the result back to the requester.

SPAs control and organize RAs. The detailed function can be divided into three categories: (1) accepts request from BAs, and fulfill them by releasing the request to RAs or subcontract it to other SPAs. (2) communicates with RAs and manage the available resources. (3) contacts with BAs proactively to advertise the available resources.

BAs are responsible for composing and providing a single virtualized service to cloud consumer agents. BAs first receive consumer requirements from CAs, and then they contact a few SPAs and makes contracts with them. When SPAs are done, BAs will return the results of available resources to CAs. BAs act as an intermediate between SPAs and CAs.

CAs accept consumer requirements from cloud consumers and translate them through service ontology. Then map the translated requirements to corresponding BAs. After the BAs return the requested results, CAs will calculate a most efficient plan and send it back to the cloud consumers.

The above-mentioned agents use Service Capability Tables (SCTs) to perform service discovery. SCTs are a variation of acquaintance network [8] while differs in adding the status of cloud agents into records. The original acquaintance network only stores the service capabilities of agents in the system. Literally, SCTs store the service capability of each agent and constantly update the status of the service. The

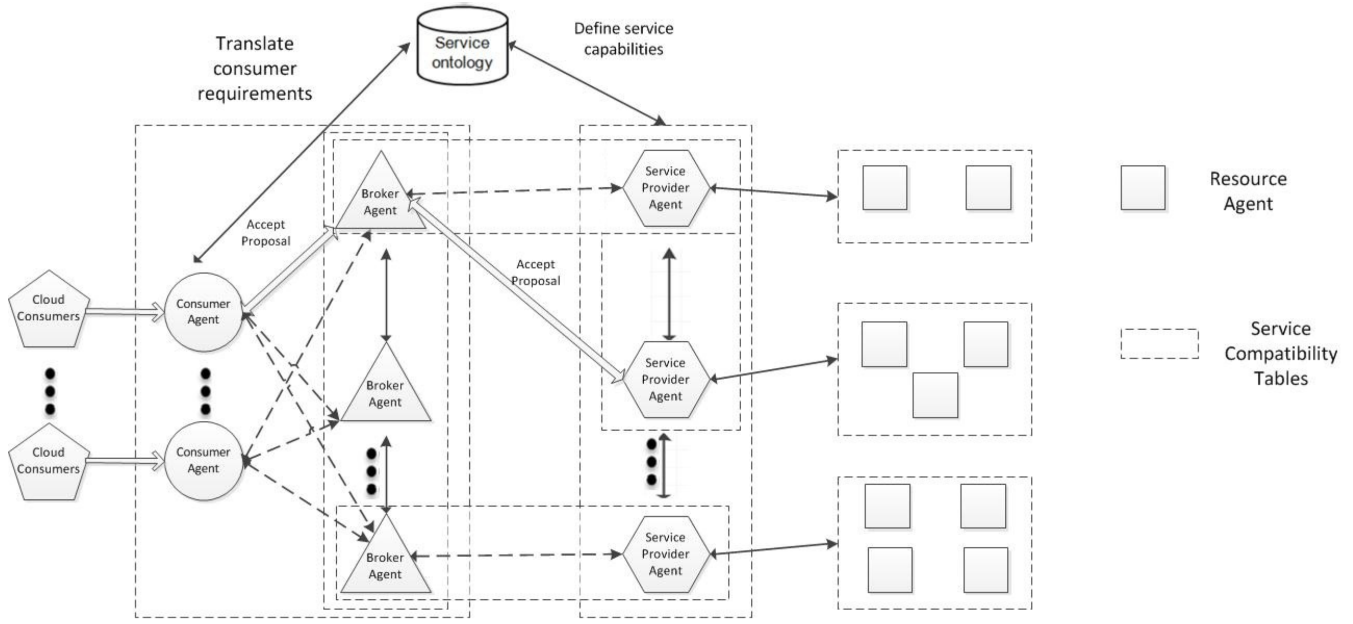
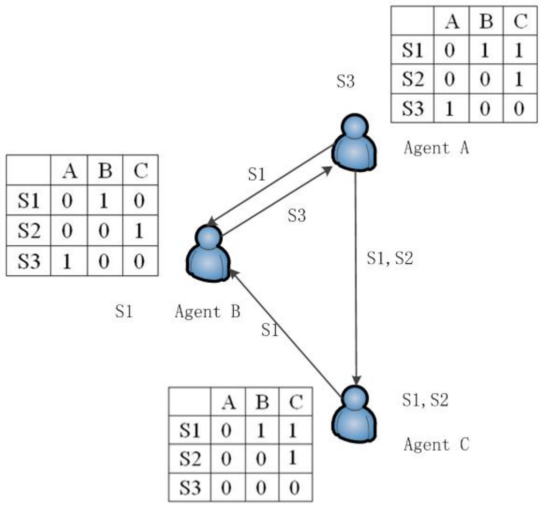


Fig. (1). Architecture of agent-based cloud computing system.

records in SCTs employ the idea of acquaintance network to store a list of acquainted agents and their service capabilities. Fig. (**2**) is an example of the acquaintance network. Each CA maintains one SCT that records a set of BAs. Each BA maintains two SCTs that record information about SPAs and other BAs.



	A	B	C
S1	0	1	1
S2	0	0	1
S3	1	0	0

	A	B	C
S1	0	1	0
S2	0	0	1
S3	1	0	0

	A	B	C
S1	0	1	1
S2	0	0	1
S3	0	0	0

Fig. (2). Demonstration of acquaintance network.

Each SPA maintains two SCTs that record information about RAs and other SPAs. Each RA maintains one SCT about their siblings under the same jurisdiction of the SPA.

From the definition of SCTs and acquaintance network, we discovered there are several drawbacks:

1). The limited entries in SCTs make it difficult for a thorough search in the whole cloud computing network topology. It will increase a great deal of overheads by increasing the entries in SCT, because the time complexity for locating corresponding entry by going through each entry one by one is *O(n)*.

2). The SCTs are unable to handle the joining and disconnection of agents. Because cloud services can be intermittent and dynamic, adopting the acquaintance network to handle service discovery will make newly joined agents without any acquaintance. Then the newly joined agents will be left alone indefinitely. Because of the intermittent and dynamic cloud computing environment, a common scenario is that some agents happened to be the entries of other agents simultaneously in the system, which is very likely to happen because of cloud computing. As a result this agent will not be able to perform effective search since there is not enough entries in the SCTs.

3). The message exchanged by agents to notify the service status can cause flood in the network. As mentioned above, SCT contains a column of service status which is constantly updated. There will be a flood in the network if some services frequently update status.

To address the above-mentioned problems, we proposed a novel service discovery approach. Experiment result show that our approach has better performance than the original SCTs based approach in the aspect of data availability in the agent-based computing environment.

This paper is organized as follows. Section 2 depicts the related works in this domain. Section 3 describes the architecture of our approach. Section 4 introduces the routing protocol for service discovery of our approach. The experimental results are in Section 5 and the discussion is given. Finally, we give our conclusion and outlines future work in Section 6.

2. RELATED WORKS

Existing research efforts in the service discovery can be divided into two categories: centralized and decentralized.

The difference between centralized and decentralized service discovery is the existence of a central node for serv-

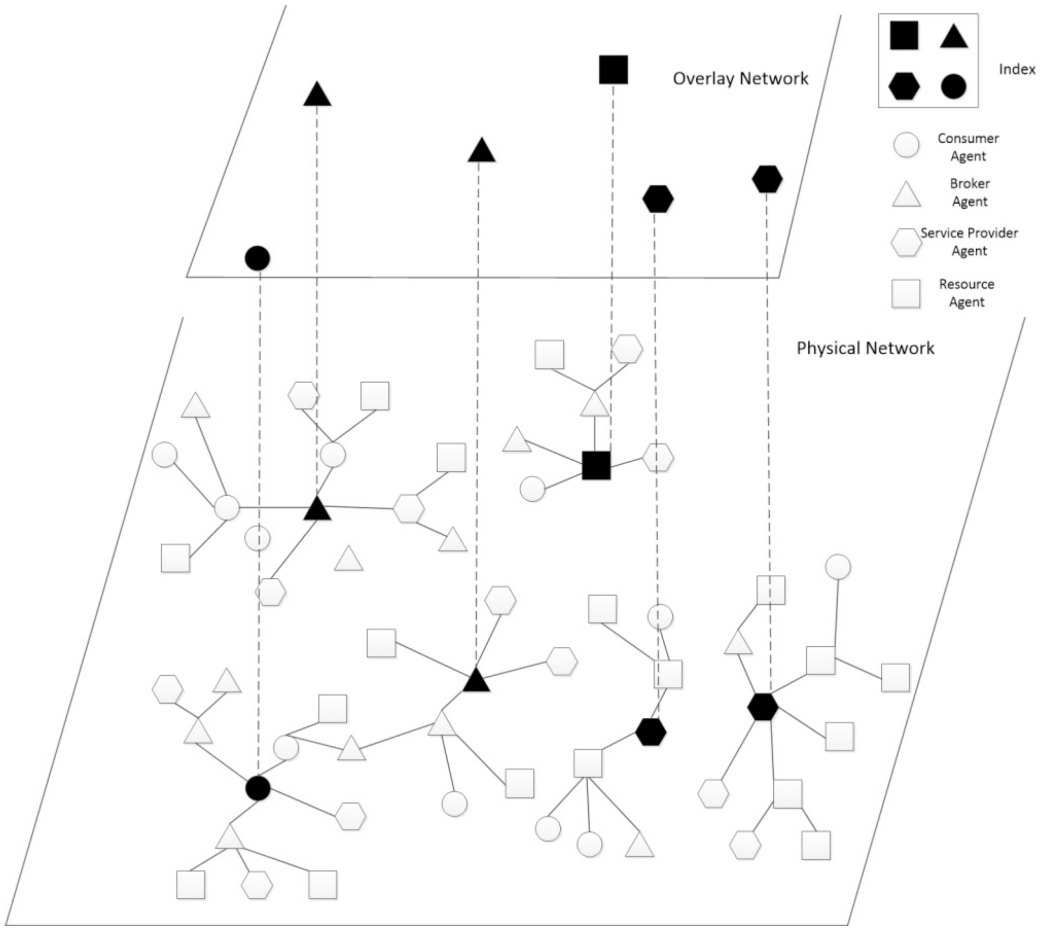


Fig. (3). Design of discovery architecture.

ice registering and discovering. Universal Description, Discovery and Integration (UDDI) [9], the most popular service discovery protocol in the last decades, is developed by IBM according to the centralized pattern. In this model, service discovery is executed by a client/server node. UDDI relies on a central server to store all the service information, service providers register their services on the central server using a unicast message. A client can retrieve service description by contacting the central server and use the service description to establish contact with the service provider. Scalability and efficiency will be the bottleneck of the single central server in the entire UDDI like centralized service discovery pattern. In the cloud computing environment with massive data and services, the scalability would be a tough problem central server. As in big data environment, massive amount of services emerge every day. It requires for an unimaginable speed of server expansion if we still use centralized service discovery approach. Then the distributed UDDI registries have been proposed to neutralize this problem. In [10], a distributed central server approach is employed to achieve scalable service discovery for mobile Ad hoc network (MANET). In [11], P. Castro *et al.* proposed a schema to connect pervasive computing device in local network using central node. In [12], M. Denny *et al.* introduced the Mobiscope which is a service discovery protocol for mobile devices.

The decentralized service discovery protocols emerged around the millennium appear to be better solutions since clients and service providers tend to discovery each other directly [10, 11]. By taking the central server out of the system the bottleneck disappeared. The Universal Plug and Play (UPnP) framework [12] is developed by Microsoft and UPnP is one of the earliest decentralized service discovery protocol putting into business use. Using the Simple Service Discovery Protocol (SSDP), clients and service providers can discovery each other directly. Service providers in UPnP send advertisement in the form of multicast ssdp:alive messages. Clients within the advertising range will be aware of the service provider and can contact service provider directly if needed. Clients can also send ssdp:discover messages to discover potential service providers nearby. UPnP can only be applied in small scale network such as home network and LAN, because the multicast messages can cause flood without configured in limited range.

Chord [13] is a structured peer-to-peer network protocol, which has been used to facilitate decentralized web service discovery. In [14], Q.He *et al.* proposed a Chord4S - a modified service discovery protocol based on Chord. Chord4S distributes the descriptions of functionally equivalent services to improve data availability. In order to improve routing efficiency, Chord4S modified the routing protocol to be sequential for locating multiple functionally equivalent services. In [15] Li, Yin, *et al.* proposed the PSWD, a distributed web service discovery protocol based on a modified version of Chord algorithm called XChord. In the original design of Chord, it does not support querying resource by XML, but author extended Chord with XML to form XChord, which is completely capable of express service requests with XML and query service description using XML.

In [16], M. Parhi *et al.* proposed a framework for service description and discovery using ontology based on multi-agent cloud environment. Because of the non-standardized specification terminology used by different cloud providers, the traditional search engines such as Google, bing, etc are not able to perform the service discovery in an effective manner. Adopting the framework proposed by M. Parhi, service description is standardized and service discovery is more accurate and efficient.

In [17], Y.-S. Chang *et al.* proposed a layered framework for service discovery on cloud environment integrated with intelligent agent. In [18], T. Uchibayashi discovered a multi-agent framework for IaaS service discovery.

In the multi-agent cloud computing environment, decentralized service discovery approaches are not appropriate. With unstructured network topology design, it will cause information flood and scalability problem. And with structured network topology like the above-mentioned Chord, the web service will have to be relocated because of the adoption of hash function in these protocols. It is similar to the work presented in [19], [but our work focused on applying the distributed centralized service discovery approach onto multi-agent cloud environment. The proposed approach supports global search with a better performance, as detailed in Section 3, 4, and 5].

3. NETWORK DESIGN OF DISCOVERY ARCHITECTURE

In this section, we will introduce the basic design of our approach and its advantage over the SCTs.

In the physical network of cloud computing, every agent is interconnected like a giant web. Fig. (**3**) illustrates the design of our approach in a network topology view. In our approach, an index is an agent acts as a central control center for the agents under its jurisdiction. The indexes are deployed under the rules that: At least one index is reachable in a fixed number of hops *n* (*n* depends on the agent density). Indexes are elected in an impartial way in order not to affect the load balance of the original physical network. The election procedure for indexes will be detailed in the next section. Indexes contain the description of web services provided by the agents under its jurisdiction and the latest service status. Other agents that are not indexed will not be responsible for storing the service descriptions. On top of the physical network, the indexes form an overlay network. Overlay network is essential for the purpose of global searching.

An agent can simply send a service lookup request to the according index when it tends to discover a service. The index will look up locally stored information to check if there is a matching service for the required one. The matching service's status should be available. If it exists in its cache, index will forward the detailed information of agent which provides service back to the agent requesting the service. If not, index will perform a global searching and send the service request to the index which most likely stores the requesting service. (Detailed service discovery techniques will be discussed in the next section). Destination index to forward the service request is based on the profiling exchanged between indexes (the profiling procedure will be discussed in the next section). The index profiling provides a brief introduction of the services it provides and the capacity it may host.

Improvements of our approach over SCT as follows:

(1) We propose an overlay network above the physical network in order to perform global searching.

(2) We adopt indexes in our approach to be able to handle intermittent and volatile network environment like cloud computing environment.

(3) Our approach generates minimize traffic during the service publication; service discovery process because of each index handles its own jurisdiction.

4. SERVICE DISCOVERY PROTOCOL

In this section, we mainly discuss the service discovery protocol we designed for the approach. It consists of two parts: Local Service Discovery and Global Service Discovery.

4.1. Local Service Discovery

As mentioned in previous sections, indexes have three functionalities: (1) Storing the service description and updating the service status of available service within the index's own jurisdiction. (2) Responding to service queries according to the stored cache. (3) Periodically broadcasting messages to the jurisdiction for presenting its existence.

An index stores service advertisements broadcasted by agents which provide certain kind of service to the index over *n* hops (*n* is the cover range of this specific index). The cache information stored in index is managed using Least Recently Used (LRU) algorithm, which replaces the least recently used item with the newest item. In this way, index can ensure the hit rate judging from Pareto principle (also known as the 80–20 rule, states that, for many events, roughly 80% of the effects come from 20% of the causes). Agents providing services will set their broadcasting time gap according to: (1) the number of agents under the same jurisdiction, by analyzing the advisement received from index or the local routing table to find out the number of nearby agents. (2) the load on the network, which can be detected from the acknowledges received from a few last sent messages. Using the two techniques above-mentioned we ensure that our approach maintains relatively low number of messages exchanged in the service discovery process.

Upon receiving the service query from an agent under its jurisdiction, the index will check locally stored information to identify whether the requested service exists in its jurisdiction. If the index finds a match or several matches, it will run a selection algorithm to decide which match to send based on the following principles: (1) Choose best match according to the similarity between requested service description and matched service using service ontology analysis. (2) Choose the most available service by going through matched service's status. If there are not any matches in the local jurisdiction, the index will perform a global searching for the requested service which will be discussed in the next section.

4.1.1. Index Broadcasting and Electing Procedures

As mentioned above, indexes are responsible for being recognized by agents which provide services and request services within their own jurisdiction. To this end, index must broadcast their existence through *n* hops. However, broadcasting may cause flood in the network. In order to control the number of messages in broadcasting, we adopt following techniques: *t* messages are broadcasted to the neighbors of the initial index only. Then out of the nodes received the broadcast message, it selects the minimum number of nodes that can reach all the nodes in their direct connected neighbors while these zones do not intersect. Then the selected nodes will forward the received messages to its neighbors, and this procedure will repeat several times. After *n* hops, the local index will be recognized by all the agents in their own jurisdiction.

With the intention of not interfering the original layout of agents, we propose an election procedure. This procedure is fair for all agents and can keep the physical network stable with unnoticeable message exchanging overhead. In order to achieve less duplication, existing indexes are also required to perform the election procedure when receiving an election request. An agent can belong to more than one index, because of the instability nature of cloud computing environment. As noted above, indexes send out presence advertisements in a time interval of *x*. Thus, when an agent does not receive any index presence advertisements in the time interval of $a \times x$ (*a* is the network load parameter depending on the current network condition), the agent begins to initiate an election process for finding an index.

By starting the election process, an agent broadcasts the election initiating message over *n* hops. Other agents within the *n* hops range receiving the election initiating message can either accept or refuse to be elected as an index. Judging from its own capability and availability, an agent can make a decision about whether being an index or not. If the agent decides it can take the responsibility by acting as an index, it will sends back an acceptance message to the election initiator with the detailed information about its capability to store cached service information, the availability of free resources and the network environment around it (the message failure rate sent into the local network, the indexes covering it, the available resource directly connect to it for instance). When the initiator receives enough acceptance messages in the time interval of *2x*, which is the time that one message can travel back and forth for *n* hops. The initiator elects the most suitable candidate to be its index by evaluating the detailed information attached to the acceptance message. The initiator values the resource around the candidate, *i.e.*, the candidate has the most neighbors and the least number of indexes it belongs to, and distance between the candidate and the initiator as two of the most important attributes. If the agent refuses to be elected as an index, it simply sends back a refuse message. The refuse messages will be discarded upon the initiator receiving them. The above mentioned election procedures ensures the election can be done efficiently with the best physical attributes indexes elected and the best distribution rate of indexes. Because one of the main factors decides which agent becomes the index is "least number of indexes it belongs to".

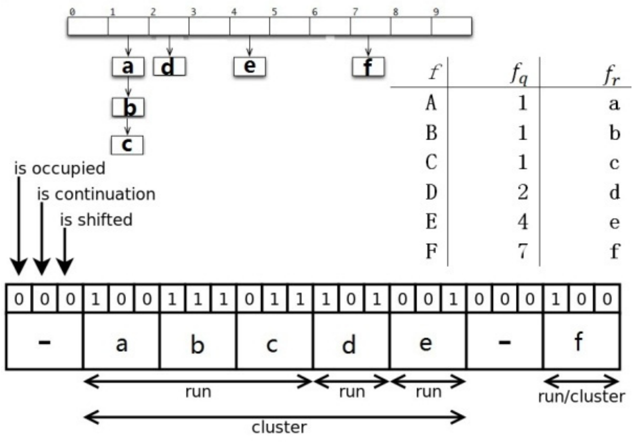
In the case of two or more agents start electing procedure at the same time, the agent with the smallest hexadecimal MAC address continues the election process while other agents awaits until the first agent finishes electing process. The above mentioned election process keeps running to adjust the network load balance from time to time. And this process ensures the intermittent network environment acts efficiently.

4.2. Global Service Discovery

The last sub section discusses the service discovery strategy within the jurisdiction. However, agents need to discover the appropriate service in the whole cloud environment. We can achieve this feature by designing a global service discovery strategy. As we mentioned above, global service discovery requires the cooperating of indexes which may store the required service. Aiming to locate the corresponding index to forward the query, we introduce the index profile that provides a brief summarization of the index resources.

4.2.1. Index Profile

We use Quotient Filter [20], a kind of approximate membership query (AMQ), to summarize the WSDL based service description of an index. An AMQ is a space-efficient probabilistic data structure used to test whether an element is a member of a set. In other words, AMQ is a dictionary that trades off space for a false positive rate on membership queries. Quotient filter is efficient in describing indexes because it does not store the whole lot of resources. It can make the large unsorted file to map each service description record into a table by adopting the hash function. The basic idea of quotient filter is to make the already stored p-bit fingerprints, which is the hash value of service description stored in index into *q* most significant bits (the quotient) and *r* least significant bits (the remainder) by employing quotienting [21].



f	f_q	f_r
A	1	a
B	1	b
C	1	c
D	2	d
E	4	e
F	7	f

Fig. (4). Example of quotient filter.

Fig. (**4**) illustrates an example of quotient filter. In the figure, we can see that a quotient filter is a compact hash table. The hash table has 2^q slots. The fingerprint f_p is the hash value of key f. We assume the quotient of f_p is f_q and the remainder be f_r. The quotient filter will try to store the

Table 1. Explanation of the bits combination.

is_occupied	*is_continuation*	*is_shifted*	**Description**
0	0	0	Empty Slot
0	0	1	This slot is the start of a run which has been shifted from its canonical slot.
0	1	0	Not used
0	1	1	This slot is the continuation of a run that has been shifted from its canonical slot.
1	0	0	This slot is the start of a run that is in its canonical slot.
1	0	1	This slot is holding start of a run that has been shifted from its canonical slot. Also the run for which this is the canonical slot exists but is shifted right.
1	1	0	Not used
1	1	1	This slot is holding continuation of a run that has been shifted from its canonical slot. Also the run for which this is the canonical slot exists but is shifted right.

remainder f_q in the slot of f_{μ}, which is called canonical slot. The quotient filter adopts linear probing as the solution of collision (if the quotients of two stored fingerprints are equal). Linear probing means if there is a collision and the canonical slot is taken then the latecomer remainders will be stored to the right of the canonical slot.

Also, we have to introduce two definitions in the quotient filter:

(1) A run is all of the remainders with the same quotient stored contiguously.

(2) A cluster is a maximal sequence of occupied slots whose first element is the only element of the cluster stored in its canonical slot. Thus, a cluster may contain one or more runs.

The additional bits in each slot have the following meaning:

(1) *is_occupied* indicates whether this slot is the canonical slot for the value stored or not.

(2) *is_continuation* indicates that whether the remainder of this slot is part of the run not.

(3) *is_shifted* indicates whether the remainder is in the canonical slot or not. Table **1** illustrates the explanation of different bits combination.

We can identify if a quotient filter contains a certain key *l* by following procedures:

(1) Get the fingerprint f_{μ} by hashing l. And get the quotient f_q and the remainder f_r by running the quotient function.

(2) If the slot f_q's three bits combination is 000 then we can determine the filter does not contain key *l*.

(3) If the slot f_q is not empty, which means the canonical slot is taken. We must locate the quotient's run in order to find which slot l is stored. We start scanning the left of f_q to locate the start of cluster, in order to locate the start of quotient's run.

(4) The scan stops when *is_shifted* is false, which indicates the start of the cluster. Then we start scanning right to skip *j* (*j* is the total number of *is_shifted* recorded when proceed (3)) runs, we compare the f_r with the records stored in the quotient's run.

(5) If matches the recores, the key *l* is in the quotient filter. Otherwise the key *l* does not exist in the quotient filter.

When two different elements map to same fingerprint we call it a false positive. The load factor of a normal hash function *α*= *n/p*, where *n* is number of elements and p = is 2^q the number of slots. The probability of a false positive is approximately $1 - e^{-\alpha/2^r} \le 2^{-r}$.

4.2.2. Service Discovery Scheme Between Indexes

A recent elected index initiates broadcasting to *n* hops while none of the indexes has been aware of. Other indexes received the message will send back profiles, which are the content of quotient filter and host service capacity. Otherwise, if an index is aware of one or more indexes from the previous electing process or from the previous knowledge when it is not an index yet, the index will send a profile request to all the known index neighbors. The index neighbors send back the profile lists requested. In one way or another, the index will create a list of index profiles when it enters the network. Later on, each index broadcasts its list of profiles to *2×n* hops, which is two times the range of its jurisdiction. In this way, indexes are able to discover each other in a relatively exchanging message saving pattern.

Every agent has at least one index according to our approach's design. Fig. (**5**) illustrates the discovery procedure.

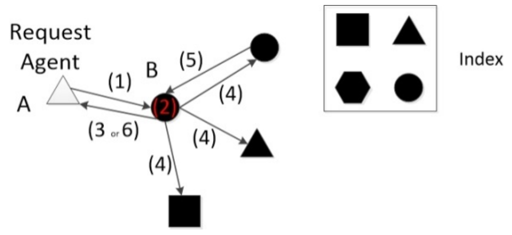
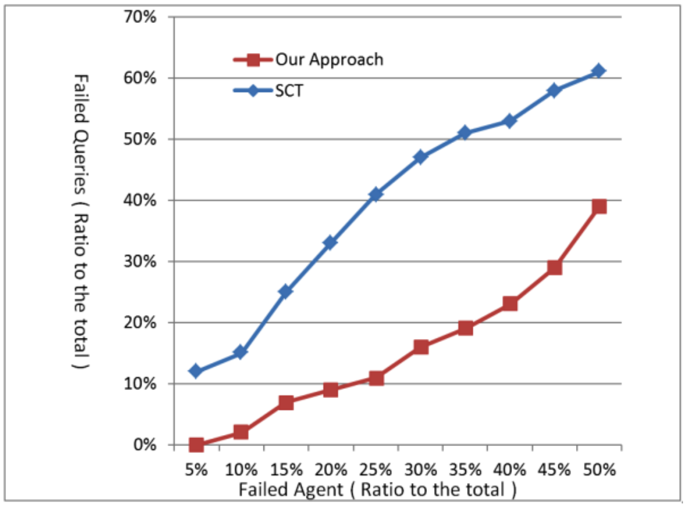
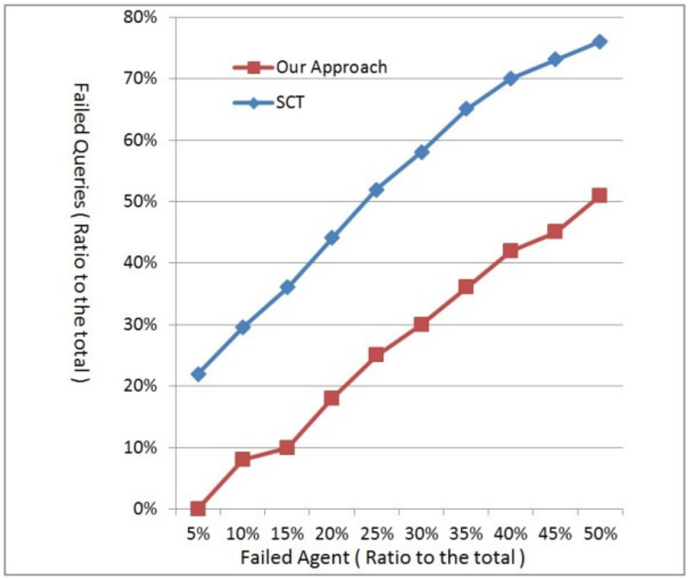


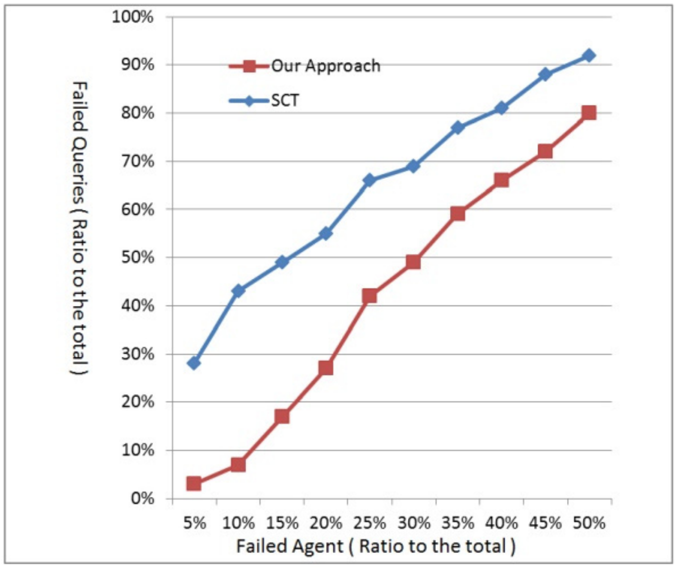
Fig. (5). Global discovery model.



(**a**) Results of the network consisting 500 agents

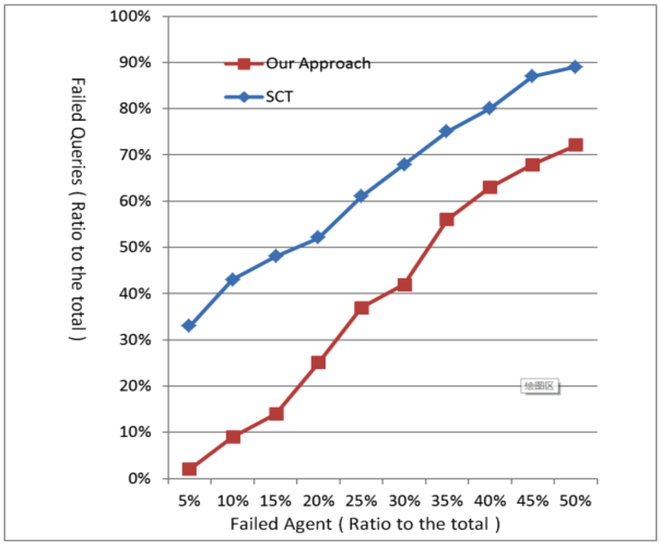


(**b**) Results of the network consisting 750 agents

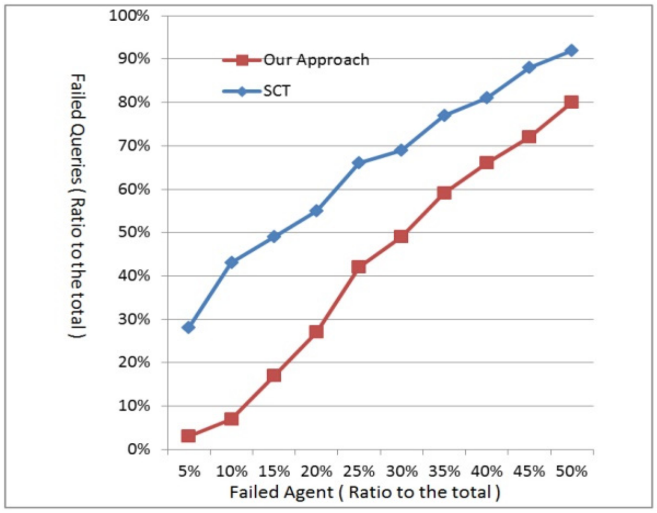


(**c**) Results of the network consisting 1000 agents

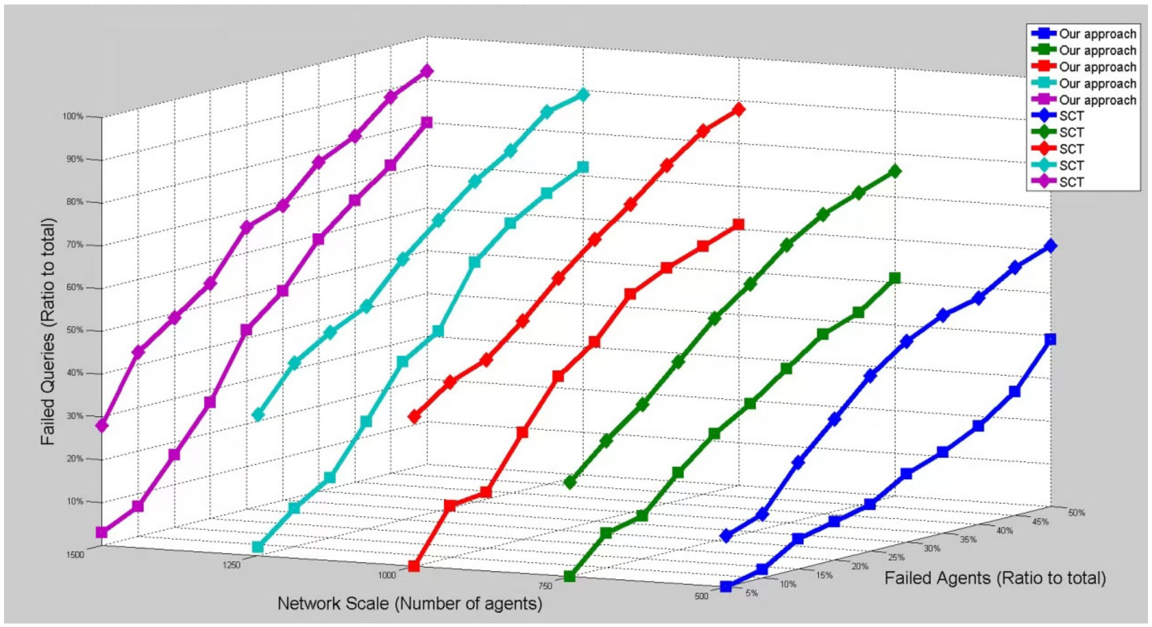
Fig. (6). Contd……



(**d**) Results of the network consisting 1250 agents



(**e**) Results of the network consisting 1500 agents



(**f**) Comparison between different sets of data

Fig. (6). Experiment results.

When an agent A searches for a specific service, the discovery procedure starts as follows:

(1) It sends a query message to a local index B.

(2) Upon receiving the query message, B starts to look into its own cache to whether the requested service is within the jurisdiction.

(3) If the service is found, B sends back the detailed service information along with the service provider's information to A. If the requested service does not exist in the local jurisdiction, B looks up its stored list of profiles. By using the quotient list query B locates all the indexes, which are most likely to have the requested service.

(4) Then B forwards the query to these indexes.

(5) Finally, requested indexes send back a match message to B.

(6) B sends a match message back to A if there exists any requested service or a no match message if the requested service does not exist neither in local jurisdiction nor other indexes. At this point, the query process is completed.

5. EXPERIMENTS

To evaluate the performance of our approach, a SCT simulation is implemented to act as a control group. Data availability is evaluated specifically because they are quite important in service discovery operations. We use OMNeT++ as a platform to execute the following experiments. The simulation is performed in the network scale of 500, 750, 1000, 1250 and 1500 agents in order to evaluate our approach's performance under different network scales. The network topology in this experiment is similar to the shown in Fig. (**3**).

In intermittent network environments, our approach has better data availability judging from the designing aspect. To simulate the intermittent network environments, we choose a portion of agents to disconnect from the network randomly, increasing from five percent to fifty percent of the total portion of agents with the pace of five percent. Then the remaining agents send out service queries randomly. The number of request queries is randomly selected from the interval [1,5]. We record the number of failed service queries to measure the data availability under intermittent network environments. We conducted the experiments under the assumption of the ratio of (consumer agents) : (broker agent) : (service provider agent) : (resource agent) is fixed to 1:1:1:1.

From Fig. (**6**), we can see that our approach has better data availability because the curve of our approach stays below the SCT curve the entire time. We observe that the failure rate of service queries is higher than the portion of failed agents. This phenomenon indicates that there is other reason except for failed agents causing query failure. The reason is that with some outdated service description in index there is a small chance it will cause query failure. To sum up, our approach has a better performance than SCT in the aspect of data availability.

6. CONCLUSION AND FUTURE WORK

Service discovery not only remains as a critical component in service oriented architecture, but also plays an important role in the cloud computing environment. This paper introduces a novel service discovery approach, which can be applied to agent-based cloud computing environment. We used the method of centralized service discovery based on service indexes after evaluating the merits and faults of it. In order to implement the features that cannot be realized in the original design of SCT and improve the data availability in the service discovery, we designed a virtual network on top of the physical network consisting of indexes. By adopting the central indexes within a small scale of jurisdiction, our approach is able to perform global service discovery with a relatively low overhead. According to experiment results, our approach improves the data availability.

In the future, we plan to build a structured network topology on the virtual network in order to achieve global service query with higher efficiency.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

This work is supported in part by National Natural Science Foundation of China (No. 61100143, 61272353, 61370128, 61428201), Program for New Century Excellent Talents in University (NCET-13-0659), Beijing Higher Education Young Elite Teacher Project (YETP0583).

REFERENCES

[1] T. Erl, R. Puttini, and Z. Mahmood, *Cloud Computing: Concepts, Technology & Architecture*, Pearson Education: US, 2013.

[2] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review,* vol. 39, pp. 50-55, 2008.

[3] D. Talia, "Cloud Computing and Software Agents: Towards Cloud Intelligent Services," In: *WOA*, *CEUR Workshop Proceedings*, pp. 2-6, 2011.

[4] K. Virani, and D. Virani, "Service composition based on multi agent in cloud environment," *International Journal of Engineering Research and Technology*, vol. 1, no. 9, 2012.

[5] K. M. Sim, "Agent-based cloud computing," *IEEE Transactions on Services Computing,* vol. 5, pp. 564-577, 2012.

[6] K. M. Sim, "Complex and concurrent negotiations for multiple interrelated e-markets," *IEEE Transactions on Cybernetics,* vol. 43, pp. 230-245, 2013.

[7] J. O. Gutierrez-Garcia, and K. M. Sim, "Agent-based cloud service composition," *Applied Intelligence,* vol. 38, pp. 436-464, 2013.

[8] J. Ferber, *Multi-agent Systems: an Introduction to Distributed Artificial Intelligence*, vol. 1: Addison-Wesley Reading: US, 1999.

[9] T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, J, Munter, and C. von Riegon, "UDDI Version 3.0," *Published Specification, Oasis,* vol. 5, pp. 16-18, 2002.

[10] E. Meshkova, J. Riihijärvi, M. Petrova, and P. Mähönen, "A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks," *Computer Networks,* vol. 52, pp. 2097-2128, 2008.

[11] C. N. Ververidis, and G. C. Polyzos, "Service discovery for mobile ad hoc networks: a survey of issues and techniques," *IEEE Communications Surveys and Tutorials,* vol. 10, pp. 30-45, 2008.

[12] B. A. Miller, T. Nixon, C. Tai, and M. D. Wood, "Home networking with universal plug and play," *IEEE Communications*

Magazine, vol. 39, pp. 104-109, 2001.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review,* vol. 31, pp. 149-160, 2001.

[14] Q. He, J. Yan, Y. Yang, R. Kowalczyk, and H. Jin, "A decentralized service discovery approach on peer-to-peer networks," *IEEE Transactions on Services Computing,* vol. 6, pp. 64-75, 2013.

[15] Y. Li, F. Zou, Z. Wu, and F. Ma, "Pwsd: a scalable web service discovery architecture based on peer-to-peer overlay network," In: *Advanced Web Technologies and Applications*, Springer: US, 2004, pp. 291-300.

[16] M. Parhi, B. K. Pattanayak, and M. R. Patra, "A multi-agent-based framework for cloud service description and discovery using ontology," In: *Intelligent Computing, Communication and Devices*, Springer: US, 2015, pp. 337-348.

[17] Y.-S. Chang, T.-Y. Juang, C.-H. Chang, and J.-S. Yen, "Integrating intelligent agent and ontology for services discovery on cloud environment," In: *Systems, Man, and Cybernetics (SMC), IEEE International Conference*, 2012, pp. 3215-3220.

[18] T. Uchibayashi, B. O. Apduhan, and N. Shiratori, "A framework of an agent-based support system for IaaS service discovery," In: *Computational Science and Its Applications (ICCSA), 13th International Conference*, 2013, pp. 28-32.

[19] F. Sailhan, and V. Issarny, "Scalable service discovery for MANET," In: *3rd IEEE International Conference on Pervasive Computing and Communications,* 2005, pp. 235-244.

[20] M. A. Bender, M. Farach-Colton, R. Johnson, R. Kraner, B. C. Kuszmaul, D. Medjedovic, P. Montes, P. Shetty, R.P. Spillane, E. Zadok, "Don't thrash: how to cache your hash on flash," *Proceedings of the VLDB Endowment,* vol. 5, pp. 1627-1637, 2012.

[21] J. Clerry, "Compact hash tables using bidirectional linear probing," *IEEE Transactions on Computers,* vol. 100, pp. 828-834, 1984.

Received: September 16, 2014 Revised: December 23, 2014 Accepted: December 31, 2014