A Decentralized Service Discovery Approach on Peer-to-Peer Networks

Qiang He, *Member*, *IEEE*, Jun Yan, Yun Yang, Ryszard Kowalczyk, and Hai Jin, *Senior Member*, *IEEE*

Abstract—Service-Oriented Computing (SOC) is emerging as a paradigm for developing distributed applications. A critical issue of utilizing SOC is to have a scalable, reliable, and robust service discovery mechanism. However, traditional service discovery methods using centralized registries can easily suffer from problems such as performance bottleneck and vulnerability to failures in large scalable service networks, thus functioning abnormally. To address these problems, this paper proposes a peer-to-peer-based decentralized service discovery approach named Chord4S. Chord4S utilizes the data distribution and lookup capabilities of the popular Chord to distribute and discover services in a decentralized manner. Data availability is further improved by distributing published descriptions of functionally equivalent services to different successor nodes that are organized into virtual segments in the Chord4S circle. Based on the service publication approach, Chord4S supports QoS-aware service discovery. Chord4S also supports service discovery with wildcard(s). In addition, the Chord routing protocol is extended to support efficient discovery of multiple services with a single query. This enables late negotiation of Service Level Agreements (SLAs) between service consumers and multiple candidate service providers. The experimental evaluation shows that Chord4S achieves higher data availability and provides efficient query with reasonable overhead.

Index Terms—Web-based services, search process

1 Introduction

Service-oriented Computing (SOC) is emerging as a new paradigm for developing distributed applications. Service discovery, among the most fundamental elements of SOC, is critical to the success of SOC as a whole. Traditional service discovery approaches of the web services technology are based on Universal Description, Discovery, and Integration (UDDI) [10]. However, centralized service registries used by UDDI may easily suffer from problems such as performance bottleneck and vulnerability to failures as the number of service consumers and requests increase in an open SOC environment. This inherent disadvantage prevents web services from being applied in large scalable service networks. As SOC environment is largely distributed, a decentralized approach appears to be the most natural way to address the above issues and achieve scalable, reliable and robust service discovery.

The Peer-to-Peer (P2P) technology provides a universal approach to improving reliability, scalability, and robustness of distributed systems by removing centralized infrastructures. In areas such as file sharing [26], Voice over Internet Protocol (VoIP) [6], [25] and video streaming [32], [33], P2P has achieved great success. Very recently, innovative research has also been carried out in the SOC field to leverage P2P computing and web services for improved service discovery. In particular, structured P2P systems such as Chord [29], CAN [21], Pastry [24], and Tapstry [35], have some characteristics that are suitable for facilitating efficient decentralized service discovery. Based on Distributed Hashing Table (DHT), structured P2P systems can achieve even data distribution and efficient query routing by controlling the topology and imposing constraints on the data distribution.

Naturally, a P2P-based decentralized service discovery approach consists of a set of distributed nodes that form a structured P2P overlay network. Upon registration, the description of a service is distributed to a relevant node to be stored in its repository. A service query can be submitted to any node, and this node, if does not store the required service description, is able to route the query to an appropriate node for resolution. Descriptions of matched services are retrieved and returned to the service consumer as the result of the query.

Although structured P2P can potentially improve the scalability of service discovery, directly applying DHT-based P2P approaches to decentralized service discovery may be weak in guaranteeing the availability of published service descriptions. This is because DHT-based systems often distribute descriptions of functionally equivalent services to the same successor node, as they have the same or similar hashing values. If such a node fails, a service consumer will not be able to discover any of these services. This disadvantage may result in serious problems in open and dynamic SOC environments where unexpected failure of nodes cannot be avoided.

- *Q. He is with the Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne 3122, Australia and the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: qhe@swin.edu.au.*
- *J. Yan is with the School of Information Systems and Technology, University of Wollongong, Australia 2522. E-mail: jyan@uow.edu.au.*
- *Y. Yang and R. Kowalczyk are with the Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne 3122, Australia. E-mail: {yyang, rkowalczyk}@swin.edu.au.*
- *H. Jin is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China. E-mail: hjin@hust.edu.cn.*

Manuscript received 6 Jan. 2009; revised 9 July 2009; accepted 18 May 2011; published online 14 June 2011.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSC-2009-01-0001.
Digital Object Identifier no. 10.1109/TSC.2011.31.

This paper proposes Chord4S, a Chord-based decentralized service discovery approach that supports service description distribution and discovery in a P2P manner. Chord is selected because it is well recognized for its flexibility and scalability and is considered suitable in large-scale SOC environments. Chord4S takes advantages of the basic principles of Chord for nodes organization, data distribution and query routing. The main aim of designing Chord4S is to largely improve the availability of service descriptions in volatile environments by distributing descriptions of functionally equivalent services to different successor nodes. In case one node fails, a service consumer is still able to find functionally equivalent services that are stored at other successor nodes. Another two features of Chord4S are to support service discovery with wildcard(s) and QoS awareness. Furthermore, Chord4S extends Chord's original routing protocol to support discovery of multiple functionally equivalent services at different successor nodes with one query, which is necessary for negotiation of a Service Level Agreement and selection of optimal service providers [5].

The rest of the paper is organized as follows. Section 2 introduces major related work. Section 3 presents the unique service description of Chord4S. Then, Section 4 addresses the service publication approach. After that, the new routing protocol of Chord4S for service discovery is proposed in Section 5, followed by discussion of experimental results in Section 6. Finally, Section 7 summarizes the major contribution of this paper and outlines future work.

2 Related Work

Our research aims at providing a scalable, reliable, and robust approach for web service discovery. An overview of web service discovery can be found in [3].

2.1 Centralized Service Discovery

The centralized client/server model has been adopted for service discovery since SOC emerged. UDDI [10] has been recognized as the most popular discovery mechanism for web services. At present, several software vendors have included UDDI support as a key feature of their software products to provide comprehensive solution for application and service integration challenges. The software includes Windows Server 2003 from Microsoft, WebSphere Studio from IBM, Oracle Enterprise Manager from Oracle, SAP Web Application Server from SAP, etc. However, as briefly discussed in Section 1, centralized infrastructures inherently suffer from poor performance in an open SOC environment that demands high scalability. Measurements have been taken to tackle the problem by employing distributed UDDI registries. In [23], Rompothong and Senivongse propose a federation of UDDI registries to enlarge the search space for service queries. Although a UDDI Federation Agent is added as an extension to a standard UDDI registry to forward queries to other federating nodes, the authors did not provide any experimental evaluation. Wu et al. [31] describe an interoperable model of distributed UDDI which divides UDDI servers into three types: root server, super domain server, and normal server. The authors adopt the philosophy of Domain Name System (DNS). Super domain servers, managed by a root server, are used to maintain normal servers. Since the model imitates DNS, it is still exposed to the same threats that DNS faces, e.g., Distributed Denial of Service (DDoS) attack. A Web Service Crawler Engine is proposed to address the performance issue caused by employing an enormous number of UDDI registries [4]. The engine crawls accessible UDDI registries and collects information in a centralized repository via which service consumers can efficiently discover required web services. The proposed approach does improve the efficiency of web service discovery by pooling distributed information, as demonstrated by provided experimental results. However, it jumps back to the issue of reliability caused by single point failure.

2.2 Decentralized Service Discovery

Decentralized service discovery is considered as a promising approach to addressing the problems caused by centralized infrastructures. In particular, some preliminary research has been conducted to utilize P2P computing for service discovery. To name a few, Zhou et al. [36] present ServiceIndex, an enhanced Skip Graph using WSDL-S as the semantic description language. Semantic attributes of web services are extracted as indexing keys to build the Skip Graph. To balance the load on peer nodes, a multilayer P2P overlay network is constructed to aggregate similar indexing keys. Similar keys are inserted into the same ServiceBag to enhance the ServiceIndex. In this way, the loss of a ServiceBag will lead to the missing of all the keys in the ServiceBag, severely jeopardizing the overall availability of the keys. Web Services Dynamic Discovery (WS-Discovery) [7], a multicast discovery protocol to locate services on a local network, is developed by BEP Systems, Canon, Intel, Microsoft and WebMethods. In WS-Discovery, a client sends a request to the corresponding multicast group to locate a target service. A proxy-specific protocol is also defined and can be switched on if a discovery proxy is available on the network. WS-Discovery is becoming popular and is already being used by some software vendors, such as the "People Near Me" contact location system in Microsoft's Windows Vista operating system. Yet WS-Discovery is tailored for ad hoc networks and there is no successful experience in applying WS-Discovery in large-scale SOC environments. Sapkota et al. [27] propose a distributed web service discovery architecture based on the concept of distributed shared space and intelligent search among a subset of spaces. In its current implementation, the shared space—the core of the architecture—is still centralized and no experimental evaluation is provided to evaluate the proposed architecture.

Chord has been used to facilitate decentralized web service discovery [11], [14], [18], [28]. Emekçi et al. [11] present a P2P framework based on Chord for web service discovery which uses finite automata to represent web services. A scalable reputation model is incorporated to rank web services based on both trust and service quality. In [14], Hu and Seneviratne propose that service providers themselves should take the responsibility to maintain their own service descriptions in a decentralized environment. Based on this concept, a decentralized service directory infrastructure is built with hashing descriptive strings into the identifiers. By doing so, peer nodes are grouped by service categories to form islands on the Chord ring. Island

Table and Native Table are created on every peer node to handle routing across islands and within islands respectively. Li et al. [18] present PSWD, a distributed web service discovery architecture based on an extended Chord algorithm called XChord. PSWD uses XML to describe web service descriptions and to express the service requests. The basic P2P routing algorithm of Chord is extended with XML to enable XML-based complicated queries. Schmidt and Parashar [28] describe a system that implements an Internet-scale DHT. The system supports searches using keywords, partial keywords and wildcard(s). To preserve the locality while mapping date elements to the index space, the system uses recursive, self-similar Space Filling Curves (SFCs). M-Chord [20] utilizes iDistance [15] to transform the metric search problem into the interval search problem in one dimension. It provides Chord with the ability to perform metric-based similarity search. We observe that none of the above has addressed the issue of data availability in open and volatile SOC environments. Node failures would lead to severe data loss when the above approaches are adopted to facilitate service discovery because descriptions of functionally equivalent services would be stored at the same successor nodes.

The research reported in this paper is similar to the work presented in [14] in using layered service identifiers to control the distribution of service descriptions. However, this research addresses the issue of data availability in open and volatile SOC environments. The proposed approach supports efficient QoS-aware service discovery and service discovery with wildcard(s), as detailed in Sections 3, 4, and 5.

3 Service Description

The service description supported by Chord4S consists of three main parts: service identifier, QoS specification, and syntax specification. The service identifiers are the identifications of the services as the basis for routing query messages. The QoS specification specifies the quality of the service that the service provider can offer. The syntax specification describes the syntax of the service, e.g., the names and data types of the input and output parameters. This paper only focuses on the former two concepts, i.e., service identifier and QoS specification. The syntax specification is usually used during the invocation of service which is beyond the scope of this paper, and hence is not addressed.

3.1 Service Identifier

Description and categorization of services can be based on either taxonomies, e.g., UNSPSC—United Nations Standard Products and Service Codes [2] and NAICS—North American Industry Classification System [1] or semantics, e.g., OWL—Web Ontology Language [8] and WSML—Web Service Modeling Language [22]. Unfortunately, none of them has been approved as general commercial or industrial standards. Chord4S supports distribution and query for hierarchical service description, e.g., "Booking.Hotel.America.USA.Texas.Huston" and "Multimedia. Video.Encoder.AVI2RM." The number and the order of the layers are application specific and can be determined by the designers of the applications. Based on this hierarchical service description, a service identifier in Chord4S is divided into two parts, function bits and provider bits. The former is used to refer to the functionality of the service while the latter is utilized to describe provider specific information. When hashing a service description to generate the service identifier, Chord4S allocates certain bits of a service identifier for the function descriptions and the rest for provider bits. A sample of service identifier consisting of five layers is presented in Fig. 1. The function bits are used for the functional service matchmaking in service discovery. The main function of the provider bits is to distinguish and distribute functionally equivalent services. Using SHA-1, one of the general consistent hashing functions, the probability of hashing two service descriptions to a same value is negligible as long as the length of the provider bits is large enough.

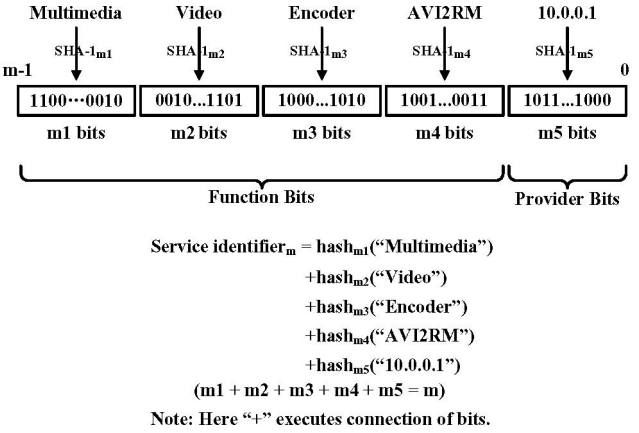


Fig. 1. Service identifier generated from hierarchical service description.

Identifiers generated from functionally equivalent services differ from each other in a certain number of the lowest bits, i.e., the provider bits. Therefore ideally 2^m (m being the length of provider bits) functionally equivalent services will yield 2^m consecutive service identifiers. Note that in Chord4S the identifiers are organized into a circle in an ascending order. Therefore, descriptions of the functionally equivalent services will be distributed to successor nodes adjacent to each other within a certain virtual segment of the identifier circle. From a global viewpoint, a Chord4S circle can be viewed as composed by a number of virtual segments, each of which contains service identifiers from a group of functionally equivalent services. With the virtual segments, the distribution of service descriptions is even because SHA-1 is applied.

Chord4S allows service descriptions in mixed structures. For example, in an application with a maximum of five-layered (four layers for functional bits and one layer for provider bits) service description, a service description like "Multimedia.Video.AVIPlayer" is also acceptable. When generating service identifier for this service description, the first three layers of the service identifier will be generated by using hash_{m1} ("Multimedia"), hash_{m2} ("Video"), and hash_{m3} ("AVIPlayer"). The fourth layer would be zero by default. In this case, the service description will be placed in a virtual segment containing all the service descriptions starting with "Multimedia.Video." A simplified Chord4S circle is presented in Fig. 2 to illustrate the specific situation.

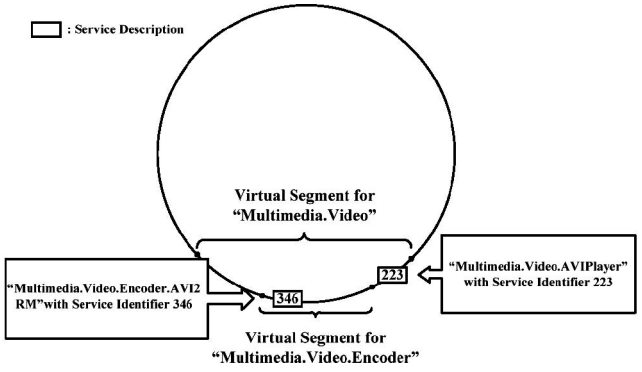


Fig. 2. Virtual segments for "Multimedia.Video" and "Multimedia.Video.Encoder."

3.2 QoS Specification

QoS awareness is a critical issue in an SOC environment. Service discovery approaches should take it into consideration because service consumers usually have specific QoS requirements. QoS-aware service discovery should filter out the services that cannot meet service consumer's QoS requirements and only return the ones that can.

Chord4S allows service providers to publish their services with quality specifications attached as advertisements. However, the quality specifications are not involved in the generation of service identifier. After finding a service description that matches its functional requirements according to the service identifier, the service consumer can look over the attached quality specification.

Chord4S supports three types of QoS attributes, defined as follows:

1. *Numeric QoS attributes.* A numeric QoS attribute is a QoS attribute that can be assigned with any value selected from a numeric interval. Many QoS attributes fall into this category, such as price, execution time, availability, etc. Comparison operators, e.g., $<, >, <=, >=$ etc, are often used to specify service consumer's QoS requirements of this type, such as "Price $<= \$1,000.00$" and "Availability $>= 0.95$".
2. *Boolean QoS attributes.* A Boolean QoS attribute is a QoS attribute that can be assigned with one of the two values: true and false. For example, a hotel booking service may have a QoS attribute named Cancellable that can be assigned with either true or false specifying that the booking can or cannot be cancelled. Two comparison operators, $==$ and $!=$ can be used to specify QoS requirements of this type, such as "Cancellable = = True."
3. *Enumerated type.* An enumerated type of QoS attribute is a QoS attribute that can be assigned with any of the enumerators as a value. For example, the types of an international postal service can be declared an enumerated type of QoS attribute that can be assigned with one of the three enumerators: Airmail, Registered Mail, or Express Mail. Two comparison operators, $==$ and $!=$, as well as set operators can be used to specify QoS requirements of this type, such as "Type = = Registered Mail" and "Type $\in$ {Registered Mail, Express Mail}."

When a service consumer has requirements of multiple QoS attributes, logical connectives can be used to combine individual QoS requirements. For example, a service consumer that requires an registered or express mail to be delivered at a price no more than $100.00 can specify a combination of QoS requirements as "Type ϵ {Registered Mail, Express Mail} AND Price $<= \$100.00$". In addition, conditional constructs can be used. For example, "IF Delivery Lead Time (Days) $<= 2$ THEN $\$100.00 <=$ Price $<= \$200.00$" expresses the semantics that if the product can be delivered within 2 days the acceptable price for the delivery service is between $100.00 and $200.00.

Besides QoS specifications, other service-specific information can be published by service providers, e.g. the behavior of services in the context, such as how a web service is used in a business process and how services interact with each other in a service composition scenario. This kind of information can be taken into consideration when looking up service providers in complicated applications. To name a few, BPEL and OWL-S descriptions can be converted to finite automata through several methods [9], [12], [19]. Then the results from hashing the finite automata or the Path Finite Automata (PFA) generated from the finite automata can be incorporated into the service description to enable semantic-enhanced service discovery [11].

4 Service Publication

In this section, we present the mechanisms for distributed service publication.

4.1 Traditional Approach in Chord

In Chord, data distribution is based on DHT. The basic principle is to store the data or the pointer to the data at the first node whose identifier is equal to or follows the identifier of the data in the identifier space. Chord uses SHA-1 as its hashing function to generate identifiers for the data and nodes. Chord organizes all the nodes into a circle modulo 2^{m}, with m being the length of the identifiers. Along the circle the routing of query is performed. The generic primitives used in Chord are as follows:

$$identifier_{node} = hash_{SHA-1}(IP_{node}),$$
$$identifier_{data} = hash_{SHA-1}(Description_{data}).$$

When distribution or lookup needs to be performed, the following primitives will be used:

$$put(identifier_{data}, data/pointer\ to\ data),$$
$$lookup(identifier_{data}).$$

The put function will store the data or the pointer to the data at the successor node whose identifier is equal to or follows the parameter identifier, while the lookup function will yield the IP address of the node responsible for the required identifier.

To enable decentralized service discovery, information about available services, i.e., service description, needs to be distributed at different nodes. However, DHT focuses on routing correctness and efficiency instead of data availability. Therefore, there is an issue of data availability that

prevents this model from being applied directly in an SOC environment. In most systems, services are required to be described in a uniform structure and style. In those cases, service descriptions from different service providers providing the same service may have the same content, e.g., in the form of "Multimedia.Video.Encoder.AVI2RM." When these service descriptions are hashed, the returned identifiers will be the same. Hence, these service descriptions will be stored at the same successor node. Similar to single point failure, failure of this successor node will lead to inaccessibility of all the services of "Multimedia.Video.Encoder.AVI2RM."

4.2 Service Publication in Chord4S

There are two traditional approaches to address the data availability issue discussed in Section 4.1, replication (i.e., storage of multiple copies of a service description at different nodes) [13] and redundancy (i.e., storage of redundant information along with the service description) [30], [34]. In an open SOC environment, they both have disadvantages. The replication approach leads to sophisticated maintenance for data availability. The redundancy approach requires significant change to the original service descriptions which may not be acceptable by the service providers. Both approaches may result in a considerably large burden on the system.

Chord4S improves data availability by distributing descriptions of functionally equivalent services to different nodes. In this way, a failed node would just have limited impact on data availability. A service consumer has the opportunity to locate the functionally equivalent services from those available nodes.

To guarantee the data availability of Chord4S-based systems and applications, some design specification can be taken into consideration. In this section, how to design Chord4S-based systems and applications to facilitate even service description distribution is discussed.

Consider a Chord4S-based overlay network consisting of n nodes, let the length of the service identifier be m and the maximum number of functionally equivalent services be k. The length of the provider bits x should be carefully calculated to achieve even service description distribution. Obviously, a smallest virtual segment should be capable of accommodating all the functionally equivalent services, as constraint (1) below

$$2^x \geq k - 1. \tag{1}$$

Hence,

$$x \geq \log_2(k-1). \tag{2}$$

Hashed into the identifier space, n nodes are distributed on the Chord4S circle with $\frac{2^m}{n}$ as the average distance between each other. So to accommodate k functionally equivalent services in a smallest virtual segment, the capability of the virtual segment is supposed to be $(k-1)\cdot\frac{2^m}{n}$. Then to allocate enough bits for provider bits, constraint (3) below should be satisfied

$$2^x \geq (k-1)\cdot\frac{2^m}{n}. \tag{3}$$

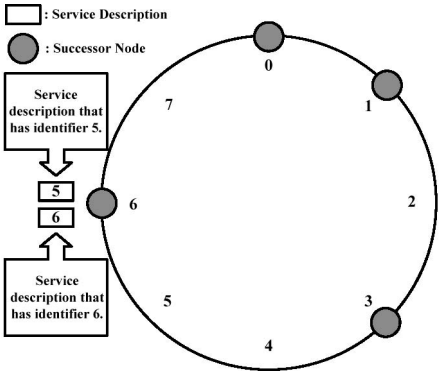


Fig. 3. Uneven distribution example.

Hence,

$$x \geq \log_2\left(\frac{k-1}{n}\cdot 2^m\right). \tag{4}$$

With constraints (2) and (4) satisfied, the descriptions of functionally equivalent services can be evenly distributed in a virtual segment which means that all of them are distributed to different successor nodes.

In certain situations, Chord4S cannot guarantee absolute even distribution. For example, some successor nodes may store more than one service of the same function. This is because that if there is no successor node with the equal node identifier to the service identifier, the service description will be stored at the successor node that has the identifier following the service identifier. Fig. 3 shows a simple sample of this situation where an identifier circle consisting of four nodes, namely 0, 1, 3, and 6. In this example, descriptions of functionally equivalent services with identifiers 5 and 6 are both stored at node 6 even their service identifiers have different contents in provider bits. When the number of nodes that joined the Chord4S circle is small, situations similar to what is presented in Fig. 3 may often occur. Consequently, the effect of data distribution may be reduced. However, the more nodes joined the system, the more effective the data distribution mechanism would be. This is also the essence of all P2P-based applications.

5 Service Query

This section presents how routing of query messages is performed in Chord4S based on the service publication approach described in Section 4.2.

5.1 Query Types

Chord4S supports two types of query: service-specific queries and queries with wildcard(s).

5.1.1 Service-Specific Query

A service-specific query contains complete details of a service description and is used to look up a specific service. In a system that allows four-layered function bits in the service descriptions, "Multimedia.Video.Encoder.AVI2RM" is a typical example of service-specific query.

To initiate a service-specific query, the service consumer needs to fill in all the layers that compose the query with explicit service information. Then each of those layers will

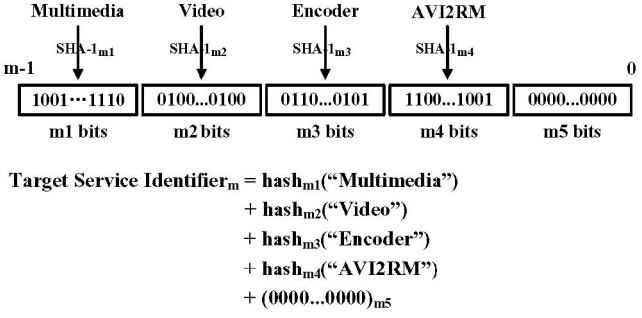


Fig. 4. Generation of target service identifier.

be hashed and the results will be connected to generate the function bits of the target service identifier. Since the objective of using service-specific queries is usually to look up a group of functionally equivalent services provided by different service providers, the provider bits of the query will be stuffed with 0s instead of being explicitly specified. The generation of target service identifier for query "Multimedia.Video.Encoder.AVI2RM" is given in Fig. 4 as an example.

5.1.2 Query with Wildcard(s)

Sometimes service consumers need to search for categories of services. For example, an amplifier service that amplifies the audio of an RM movie file can be composed using three component services which correspond to three specific steps: audio extraction, audio amplification, and video/audio combination. Therefore, the service consumer needs to find the component services from three categories: "Multimedia.Video.AudioExtractor," "Multimedia.Audio.Amplifier," and "Multimedia.Combiner," and select the ones whose inputs and outputs match. In such cases, service queries using wildcard(s) are necessary, e.g., "Multimedia.Video.AudioExtractor.*", "Multimedia. Audio.Amplifier.*", and "Multimedia.Combiner.*.*".

When solving a query with wildcard(s), it is actually looking up a virtual segment composed by nodes succeeding service descriptions that fall into the target service category. The generation of target service identifier—or more specifically target service category identifier—for a query with wildcard(s) is similar to that for a service-specific query. The difference is that the layers corresponding to the wildcard(s) will be stuffed with 0s.

5.2 Query Forwarding

5.2.1 Forwarding Service-Specific Queries

In Chord, a service consumer could easily get a list of matched service descriptions stored at the same successor node. However, the service publication approach described in Section 4.2 eliminates Chord's capability of returning multiple matched services, as the query stops at the node where the first matched service is located. In Chord4S, for a service consumer to find multiple functionally equivalent services with one query, the query must be routed across the corresponding virtual segment of the identifier circle until sufficient services required by the service consumer have been found. In this research, an improved routing protocol is designed for Chord4S, which supports further routing of a query to other nodes when it reaches a matched successor

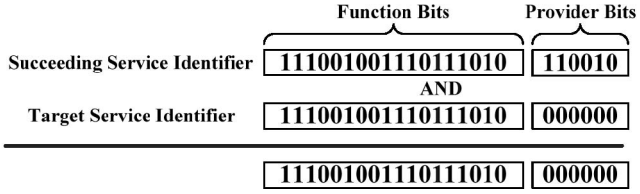


Fig. 5. Service matching operation for forwarding a service-specific query.

node. The routing performance degradation caused is reasonable where details are addressed in Section 6.

Each initiated query message contains the following basic information: a counter and a target service identifier. The target service identifier includes function bits and provider bits with the provider bits stuffed with 0's. To find out if a service matching succeeds, a node performs a binary AND operation between each of its succeeding service identifiers and the target service identifier in the query. If the result equals to the target service identifier, then the matching succeeds. Logically, this AND operation is used to extract the identifier of the virtual segment that the node belongs to and to find out if this identifier equals to the function bits in the required service identifier. A sample of matched service identifier is shown in Fig. 5. As the result from the AND operation is equal to the target service identifier, the service description that the node stores meet the service consumer's requirements.

After a matched service description is found, a query will still be passed along the circle until sufficient service providers are found. Encountered matched successor nodes must perform three tasks.

1. Get a copy of the *min(m,query.counter)* matched service descriptions it contains, with m being the number of matched service descriptions, and add it into the query message as entries of the list of candidate service providers.
2. Subtract the value of the counter by *min(m, query. counter)*.
3. Check whether the counter equals to 0. If so, send the query message back to the service consumer and the routing of this query message ends. Otherwise, route the query message to the next node according to its finger table (a routing table maintained by the Chord node).

As defined in Section 3.1, in a service identifier, the function bits are used to refer to the functionality of the service while the provider bits are used to distinguish service providers. With provider bits stuffed with 0s, the required service identifier actually represents the identifier of the virtual segment that target successor nodes belong to. Therefore, given a target virtual segment identifier, the identifiers of target successor nodes can be specified by enumerating legitimate node identifiers in the target virtual segment. For example, assuming that the service identifier length l is 10, the lengths of function bits and provider bits are 8 and 2, respectively. Hence the identifiers of the possible successor nodes that succeed required service identifiers with function bits "11000101" and provider bits "00" (i.e., decimal code: 788), which is also the virtual segment identifier, include 11000101 00

```
//when node n receives a query message
1.  n.find_successor(message)
2.      if (message.identifier <= identifier)
3.          if message.counter == 0
4.              return message;
5.          end if
6.          for i = 1 upto services.length
7.              if match_making(message, services[i])
8.                  message.counter = message.counter - 1;
9.                  message.candidate_service_provider.add
                    (services[i].descriptions);
10.                 if message.counter == 0 break
11.                 end if
12.             end if
13.         end for
14.         n' = find_next_successor(message.identifier);
15.         if n' != null
16.             return n'.find_successor(message);
17.         else return message;
18.         end if
19.     end if
20.     if (message.counter != 0)
21.         n' = closest_preceding_finger(message.identifier);
22.         return n'.find_successor(message);
23.     end if
24. end n.find_sucessor(message)

//return next closest successor node
25. n.find_next_successor(identifier)
26.     max_identifier = get_max_potential_node_identifier;
27.     min_identifier = get_min_potential_node_identifier;
28.     for i = 1 upto finger[].length
29.         if (finger[i].node ∈ (min_identifier, max_identifier))
30.             return finger[i].node;
31.         end if
32.     end for
33.     return n;
34. end n.find_next_successor(identifier)

//return closest finger preceding identifier
35. n.closest_preceding_finger(identifier)
36.     for i = finger[].length downto 1
37.         if (finger[i].node ∈ (n, identifier))
38.             return finger[i].node;
39.         end if
40.     return n;
41. end n.closest_preceding_finger(identifier)
```

Fig. 6. Pseudocode for finding successor operation.

(i.e., decimal code: 788), 11000101 01 (i.e., decimal code: 789), 11000101 10 (i.e., decimal code: 790), 11000101 11 (i.e., decimal code: 791). Therefore, when resolving a query of service "1100010100," node n needs to find successor nodes in the target virtual segment consisting of nodes 788, 789, 790, and 791.

The pseudocode that implements the service discovery process is shown in Fig. 6. When looking for matched successor nodes, node n checks the entries of its finger table to find the node with the smallest identifier in the target virtual segment which is about to take responsibility of keeping routing the query. Thus the algorithm always makes progress until sufficient service descriptions have been found. Function *closest_preceding_finger* is used to request n to find the node known by n that most closely

```
//the match making function
1.  n.match_making(message, services[j])
2.      if is_matched(message.identifier, services[j].identifier)
3.          //check if a service meets the QoS requirements
4.          for i = 1 upto message.QoS.entries.length
5.              if is_unmatched(message.QoS.entries[i],
6.                  services[j].QoS.entries[i])
7.                  return false;
8.              end if
9.          end for
11.         return true
12.     end if
13.     return false
14. end n.match_making(message, services[j])
```

Fig. 7. Function of match making.

precedes *message.id*. However, the implementation is different from Chord as a result of our novel hierarchical structure for service descriptions.

To request services with QoS constraints, service consumers include their prespecified QoS requirements in the query messages along with the target service identifier. The process of looking up services is then divided into two steps: functional and nonfunctional. At the first step, the query message is forwarded by nodes following the routing protocol. When the query message reaches a successor node that stores a matched service description, the service discovery proceeds to the second step where the successor node checks the entries of the QoS requirements one by one to see if the service meets the QoS requirements specified in the query message. Fig. 7 presents the pseudocode that implements the match making function (Line 7 in Fig. 6).

If the service meets the QoS requirements, the successor nodes first adds the corresponding service description into the list of candidate service providers in the query message and then forward the query message (or returns the query message to the service consumer if the discovery process completes). Otherwise, the query message will be simply forwarded according to the routing protocol.

Usually, after the process of service discovery, service consumers may want to negotiate with candidate service providers over the negotiable QoS attributes of the services to establish SLAs for future service provision [16], [17]. In such cases, during the process of service discovery, service consumers can specify flexible QoS requirements, e.g., $\$250.00 <= \text{Price} <= \$350.00$, instead of stringent QoS requirements, e.g., $\text{Price} == \$300.00$, to look up service providers that they may be able to strike a bargain via SLA negotiation.

5.2.2 Forwarding Queries with Wildcard(s)

The process of forwarding queries with wildcard(s) is similar to that of forwarding service-specific queries. Instead of all the function bits, only the bits generated from explicit service information will be taken into consideration. Because a query with wildcard(s) is used to look up services that belong to a specified service category, during the process of forwarding this type of queries, all the successor nodes storing service descriptions that fall into the specified service categories are considered matched. An example is

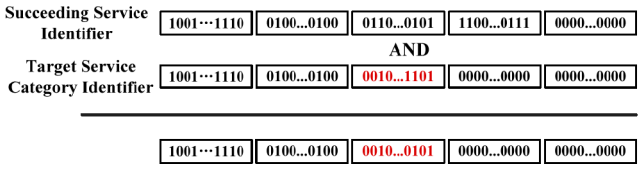


Fig. 8. Service matching operation for forwarding a service query with wildcard(s).

given in Fig. 8 to demonstrate the AND operation between a target service category identifier and an unmatched service identifier. As the result from the AND operation equals to the target service category identifier, the service descriptions that are stored at the node fall into the service category required by the service consumer.

Since a service category corresponds to a virtual segment in the Chord4S circle, to serve for a service query with wildcard(s), the corresponding virtual segment needs to be traversed. For example, to serve for the service query "Multimedia.Video.AudioExtractor.*", the query must be passed through the virtual segment "Multimedia.Video.AudioExtractor". The forwarding of query message of this type proceeds as follows:

1. Obtain the identifier of the target virtual segment by hashing the explicit part of the query.
2. Calculate the maximum node identifier in the virtual segment.
3. Look up the successor node whose identifier is equal to or follows the identifier of the target virtual segment.
4. Pass on the query through starting from the node found at step 3 until the identifier of the next node that exceeds the maximum node identifier.

An example of forwarding the query "Multimedia. Video.AudioExtractor.*" is given in Fig. 9 where all the five service descriptions that fall into the service category "Multimdeia.Video.AudioExtractor" are traversed one after another.

The implementation of forwarding queries with wildcard(s) is similar to that of forwarding service-specific queries, albeit with the matching making function (Line 7 in Fig. 6) replaced with the one that implements the matching operation exampled in Fig. 8.

5.3 Performance Analysis

The analysis follows the assumption, without losing the generality, that all service descriptions strictly conform to the structure described in Section 3.

The service discovery process in Chord4S is different from traditional approaches based on the original Chord. The main difference is that successor nodes may need to forward the query message based on their own routing information. Suppose that node d wishes to resolve a query for m successor nodes of service s. Let $p_1, p_2, \ldots, p_m$ be the successor nodes that succeed service description s, sorted by ascending node identifiers. Assume that functionally equivalent services be completely distributed at different successor nodes, i.e., each successor node can only store one matched service. In general, node p_1, the one with the lowest node identifier, will be populated at the edge of

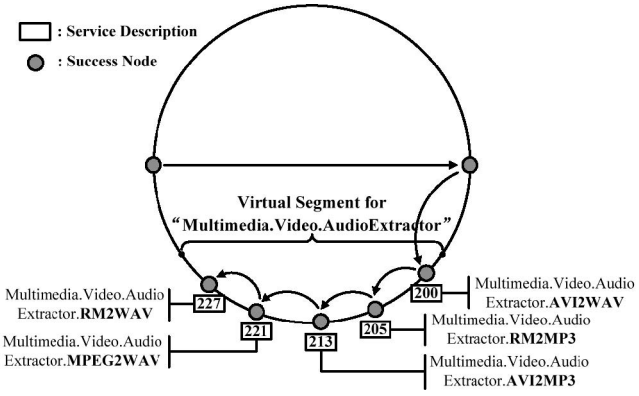


Fig. 9. Forwarding query for "Multimedia.Video.AudioExtrator.*".

the certain virtual segment that aggregates descriptions of service s. Therefore, p_1 will be the first one to be found when the query message is routed clockwise into the virtual segment. Because Chord4S is based on Chord, it inherits the desirable properties of Chord: taking $O(\log N)$ hops to find the first successor node p_1, where N is the number of nodes that constitute the system based on Chord. For a Chord4S circle that allows a maximum of five-layer service description, the maximum path length from p_1 to another arbitrary node in the same virtual segment is $2^{N/5}$, which is the maximum distance between two edge nodes of the virtual segment. To find all the other matched successor nodes, p_1 only needs to traverse the virtual segment it belongs to, which consists of at most $2^{N/5}$ nodes. Using its finger table, in the worst case, the maximum hops needed for p_1 to find the next matched successor node (if there is any) is $O(\log(N/5))$. In this case, the maximum number of hops will be $O(\log N) + O(\log(N/5)) = O(\log N)$.

When there are several matched successor nodes in p_1's finger table, i.e., $p_2, p_3, \ldots, p_m$, p_1 will route the query message directly to its closest successor, i.e., p_2. Following this, p_2 may route the query message directly to p_3 if p_3 exists. The query message will be passed along the virtual segment until sufficient services have been found. This greatly reduces the forwarding overhead. Thus, it is concluded that the total number of necessary hops is $O(\log N)$, regardless of the number of services the consumer needs to discover with one query.

For service discovery with wildcard(s), a query message has to traverse a virtual segment in order to look up the service descriptions that fall into a category. Therefore, the hops taken for a service discovery with wildcard(s) to complete consists of two parts: the hops taken to find the target virtual segment and the hops taken to traverse the target virtual segment. It is intuitive that the more wildcard(s) there are in the query, the more hops will be taken for the query to complete as the bigger the target virtual segment is the more matched successor nodes it contains that need to be traversed. For example, the query "Multimedia.Video.*.*" takes more hops than the query "Multimedia.Video.Encoder.*" to complete. The routing performance of Chord4S for service discovery with wildcard(s) also depends on the actual size of the target virtual segment, which is determined by the size of the function bits in the service identifier for the virtual segment (see Section 3.1). Bigger virtual segments can

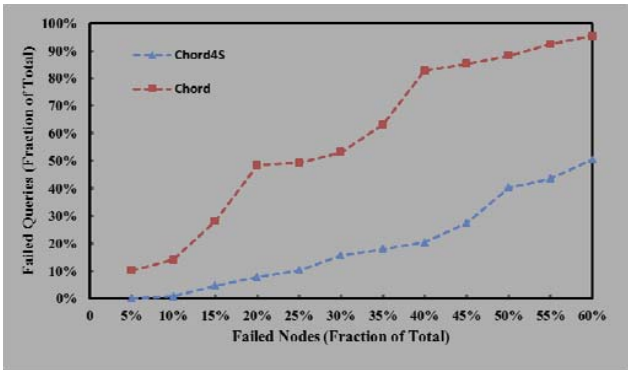
accommodate more successor nodes and hence it takes the query messages more hops to traverse the segments. Let m1, m2, m3, m4, m5 be the number of bits allocated for layers 1, 2, 3, 4, and 5 of the service identifiers, respectively, where $m1 + m2 + m3 + m4 + m5 = m$ and $N = 2^m$. When a query message with wildcard(s) is issued, first of all, it takes $O(\log N)$ hops to find the target virtual segment. Then the query message has to traverse the entire target virtual segment. The required hops depends on the size of the target virtual segment: $O(2^{m4})$, i.e., $O(N)$, for virtual segment "Multimedia.Video.Encoder," $O(2^{m3+m4})$, i.e., $O(N)$, for "Multimedia.Video," and $O(2^{m2+m3+m4})$, i.e., $O(N)$, for "Multimedia." Therefore, for service discovery with wildcard(s) to complete, the maximum hops will be $O(\log N) + O(N) = O(N)$. It is intuitive that with wildcard(s), the hops required would potentially increase linearly with the number of nodes in Chord4S in the worst case.

6 Experimental Evaluation

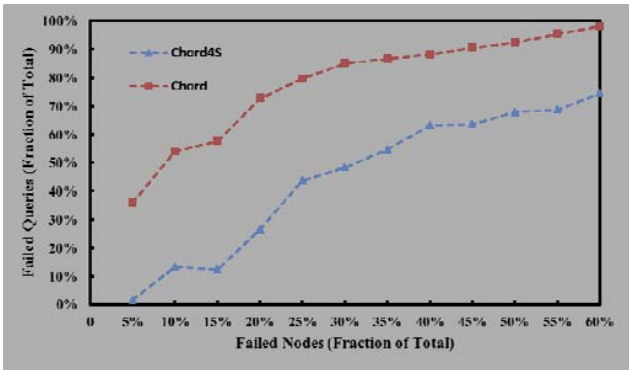
To evaluate the performance of Chord4S, a Chord simulator is extended to support Chord4S topology control, data distribution and routing protocol. As the configuration and operation of the underlying overlay network is based on Chord, Chord4S inherits good scalability with low communication cost and state maintenance cost for service discovery, as demonstrated in [18]. Data availability and routing performance were evaluated particularly because they are of great importance in Chord4S. Simulations were performed in overlay networks consisting of $2^7(128)$, $2^{11}(2{,}048)$, and $2^{15}(32{,}768)$ nodes, in order to evaluate the performance of Chord4S in environments on different scales.

6.1 Data Availability

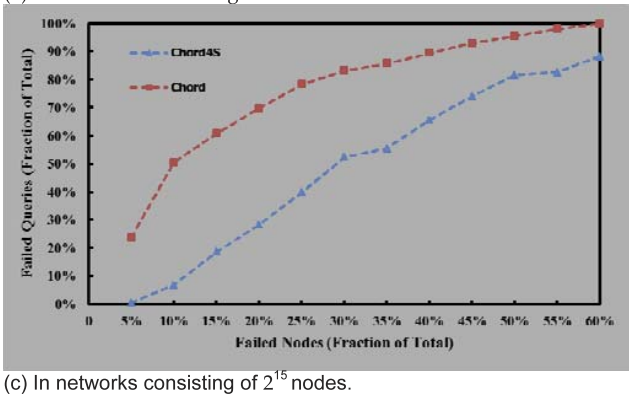
A unique feature, and also a main design goal of Chord4S is the high data availability in volatile environments. To set up the volatile environments, we randomly select a fraction of nodes participating in the network to fail in each experiment, increasing from 5 to 60 percent by steps of 5 percent. Then the remaining nodes were randomly selected to send queries for services. The number of required functionally equivalent services in each query is randomly picked from the interval [1], [16]. To evaluate the data availability, we measured the fraction of the failed queries. We conducted two sets of experiments, one with Chord and the other with Chord4S. Fig. 10 compares the results from the two sets of experiments, where the Chord4S curves always start at a much lower point and continue to stay at lower points compared to Chord. It can be observed that the fractions of failed queries are higher than the fraction of failed nodes. This result indicates that other than the service descriptions loss along the failed nodes something else also caused failed queries. The reason is that without stabilization some entries in existing nodes' finger tables became invalid. Those invalid entries yielded some failed queries and decreased the data availability in both cases of Chord and Chord4S because some queries could not be forwarded correctly. To conclude, the experimental results demonstrate that Chord4S provides better data availability than Chord in different volatile environments on different scales.



(a) In networks consisting of 2^7 nodes.



(b) In networks consisting of 2^{11} nodes.



(c) In networks consisting of 2^{15} nodes.

Fig. 10. Data availability.

6.2 Routing Performance

To evaluate the routing performance of Chord4S, the average number of hops needed for a query of a certain number of functionally equivalent services was measured. The number of required services per query is randomly picked from the interval [1], [16]. Note that when only one service is required, the discovery process equals to that of Chord and the routing completes when the first matched service is found. In the experiments, nodes were selected randomly to send queries for service. As shown in Fig. 11, the average number of hops increases proportionally to the number of required services without significant performance degradation. When a set of descriptions of functionally equivalent services are evenly distributed within a virtual segment, it often takes only one more hop to find another matched service because they are distributed next to each other. It is clear that with such reasonable extra hops, multiple functionally equivalent services can be

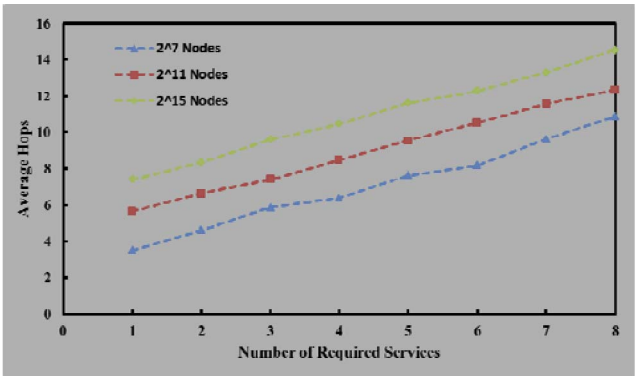


Fig. 11. Routing performance for service-specific queries.

TABLE 1
Specifics of Experiments on Service Discovery with Wildcard(s)

# of Experiments	# of Nodes	Service Identifier	Function Bits	Provider Bits
1	2^7	7	4	3
2	2^{11}	11	8	3
3	2^{15}	15	12	3

found with data availability maintained at a higher level than Chord. This feature makes Chord4S more feasible for applications in a dynamic and open SOC environment.

We also conducted experiments on the routing performance of Chord4S for service discovery with wildcard(s) and QoS awareness. Specifics of the experiments can be found in Table 1. Fig. 12 demonstrates the routing performance of Chord4S for service discovery with wildcard(s) in the experiments where service identifiers consist of 7, 11, 15 bits. The experimental results conform to the analysis presented in Section 5.3. However, considering the fact that the service descriptions are evenly distributed and a query with wildcard(s) have to traverse the entire target virtual segment, the average hops, up to 518 for a query like "Multimedia.Video.*.*" in a network consisting of 2^{15} nodes, are quite reasonable.

To assess the routing performance of Chord4S for QoS-aware service discovery, we also conducted three sets of experiments. In these experiments, the services to be requested have two QoS attributes: price and process time. The service providers' capacities for providing QoS comply with predefined normal distribution. The service consumers issued queries with different QoS requirements. The values of the required QoS attributes are randomly selected from the normal distribution used to generate the service providers' capacities for providing QoS. Based on the above experimental configuration, Figs. 13a[1] and 13b present the routing performance of Chord4S for QoS-aware service discovery. Compared to the performance of Chord4S for service discovery without QoS awareness as presented in Fig. 11, the average hops needed for QoS-aware service discovery increases by specifically 64 percent for the queries with the price requirement and 82 percent for the ones with

1. The results from the experiments with QoS requirements for process time are similar and hence are not presented.

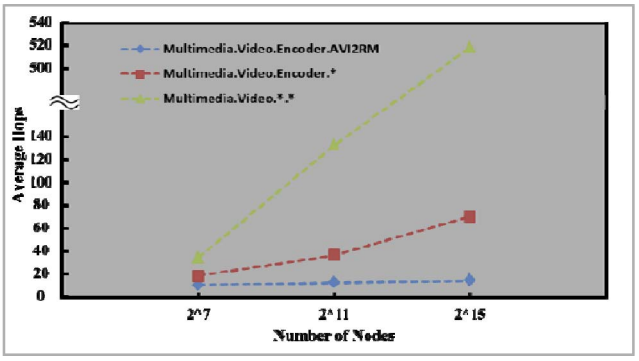
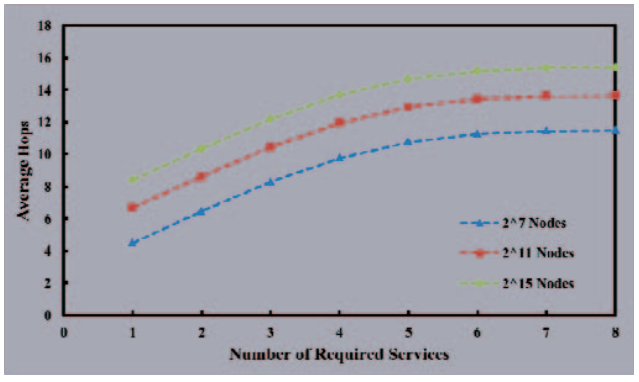
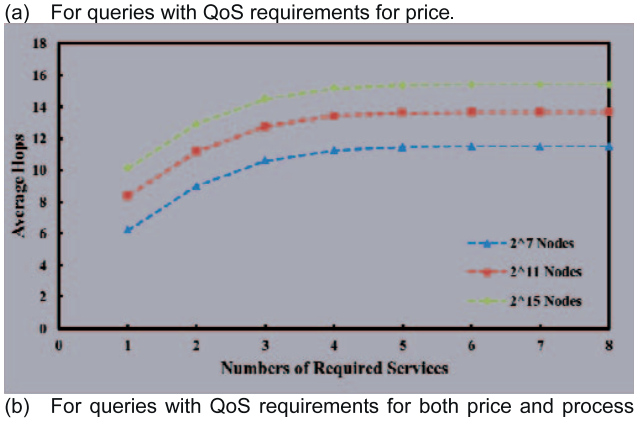


Fig. 12. Routing performance for queries with wildcard(s).



(a) For queries with QoS requirements for price.



(b) For queries with QoS requirements for both price and process time.

Fig. 13. Performance for QoS-aware service discovery.

both price and process time requirements. As the number of QoS attributes that the service consumers have requirements for increases, the average number of hops needed for Chord4S to complete routing the query messages will increase accordingly. The reason is that requirements for more QoS attributes increase the level of difficulty to find satisfactory service descriptions, and hence requires visiting more successor nodes. To conclude, the extra hops necessary for Chord4S to facilitate service discovery with QoS awareness are acceptable.

7 Conclusion and Future Work

Service discovery is a critical component in service-oriented computing. Over recent years, peer-to-peer-based service discovery has attracted researchers' attention after the deficiencies of centralized service discovery are identified. This paper has proposed Chord4S, a peer-to-peer-based

approach for decentralized service discovery. To improve data availability, Chord4S distributes the descriptions of functionally equivalent services. Chord4S supports QoS-aware service discovery and service discovery with wild-card(s). An efficient routing algorithm is developed to facilitate queries of multiple functionally equivalent services. Chord4S is scalable, reliable, and robust due to the enhanced peer-to-peer architecture. Experimental results demonstrate that Chord4S can achieve high data availability and efficient query of multiple functionally equivalent services with reasonable overhead.

In the future, integration of semantic information of services into Chord4S using popular tools, such as Petri Net and WSMO, will be investigated in order to increase the flexibility and accuracy of the service discovery.

ACKNOWLEDGMENTS

This work was partly funded by the Australian Research Council Discovery Project Scheme under grant No. DP0663841, the National Science Foundation of China under grant No. 90412010, and the ChinaGrid project from the Ministry of Education of China. The authors are grateful for the simulation implementation and English proof reading by S. Hunter. This paper is an extension of "Chord4S: A P2P-Based Decentralized Service, Discovery Approach," published in the Proceedings of the 2008 IEEE International Conference on Services Computing (SCC 2008), 8-11 July 2008, Honolulu, Hawaii.

REFERENCES

[1] "North American Industrial Classification Scheme (NAICS) codes," http://www.naics.com/, 2012.

[2] "Universal Standard Products and Services Classification (UNSPSC)," http://www.unspsc.org/, 2012.

[3] R. Ahmed, N. Limam, J. Xiao, Y. Iraqi, and R. Boutaba, "Resource and Service Discovery in Large-Scale Multi-Domain Networks," *IEEE Comm. Surveys and Tutorials,* vol. 9, no. 4, pp. 2-30, Oct.-Dec. 2007.

[4] E. Al-Masri and Q.H. Mahmoud, "Crawling Multiple UDDI Business Registries," *Proc. 16th Int'l Conf. World Wide Web (WWW '07),* pp. 1255-1256, 2007.

[5] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani, "PAWS: A Framework for Executing Adaptive Web-Service Processes," *IEEE Software,* vol. 24, no. 6, pp. 39-46, Nov./Dec. 2007.

[6] S. Baset and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," *Proc. IEEE INFOCOM,* pp. 1-11, 2006.

[7] J. Beatty, G. Kakivaya, D. Kemp, T. Kuehnel, B. Lovering, B. Roe, C. St.John, J. Schlimmer, G. Simonet, D. Walter, J. Weast, Y. Yarmosh, and P. Yendluri, "Web Services Dynamic Discovery (WS-Discovery)," http://specs.xmlsoap.org/ws/2005/04/discovery/ws-discovery.pdf, 2005.

[8] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L.A. Stein, "OWL Web Ontology Language Reference," http://www.w3.org/TR/owl-ref, 2004.

[9] Z. Cheng, M.P. Singh, and M.A. Vouk, "Verifying Constraints on Web Service Compositions," *Real World Semantic Web Applications,* June 2002.

[10] L. Clement, A. Hately, C. von Riegen, and T. Rogers, "UDDI Version 3.0.2," OASIS, http://www.uddi.org/pubs/uddi_v3.htm, 2004.

[11] F. Emekçi, O.D. Sahin, D. Agrawal, and A.E. Abbadi, "A Peer-to-Peer Framework for Web Service Discovery with Ranking," *Proc. IEEE Int'l Conf. Web Services (ICWS '04),* pp. 192-199, 2004.

[12] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-Based Verification of Web Service Compositions," *Proc. IEEE 18th Int'l Conf. Automated Software Eng. (ASE '03),* pp. 152-163, 2003.

[13] V. Gopalakrishnan, B.D. Silaghi, B. Bhattacharjee, and P.J. Keleher, "Adaptive Replication in Peer-to-Peer Systems," *Proc. 24th Int'l Conf. Distributed Computing Systems (ICDCS '04),* pp. 360-369, 2004.

[14] T.H.-T. Hu and A. Seneviratne, "Autonomic Peer-to-Peer Service Directory," *IEICE Trans. Information System,* vol. E88-D, no. 12, pp. 2630-2639, 2005.

[15] H.V. Jagadish, B.C. Ooi, K.-L.Y. Tan, and R. Cui Zhang, "iDistance: An Adaptive B+-Tree Based Indexing Method for Nearest Neighbor Search," *ACM Trans. Database Systems,* vol. 30, no. 2, pp. 364-397, 2005.

[16] L.-j. Jin, V. Machiraju, and A. Sahai, "Analysis on Service Level Agreement of Web Services," technical report, HP Laboratories, http://www.hpl.hp.co.uk/techreports/2002/HPL-2002-180.pdf. 2002.

[17] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tueck, "SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems," *Proc. Eighth Int'l Workshop Job Scheduling Strategies for Parallel Processing (JSSPP '02),* 2002.

[18] Y. Li, F. Zou, Z. Wu, and F. Ma, "PWSD: A Scalable Web Service Discovery Architecture Based on Peer-to-Peer Overlay Network," *Proc. Sixth Asia-Pacific Web Conf. Advanced Web Technologies and Applications (APWeb '04),* pp. 291-300, 2004.

[19] S. Narayanan and S.A. McIlraith, "Simulation, Verification and Automated Composition of Web Services," *Proc. 11th Int'l Conf. World Wide Web (WWW '02),* pp. 77-88, 2002.

[20] D. Novak and P. Zezula, "M-Chord: A Scalable Distributed Similarity Search Structure," *Proc. First Int'l Conf. Scalable Information Systems,* 2006.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proc. Conf. SIGCOMM,* pp. 161-172, 2001.

[22] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel, "The Web Service Modeling Language (WSML) v0.21," http://www.wsmo.org/TR/d16/d16.1/v0.21/, 2005.

[23] P. Rompothong and T. Senivongse, "A Query Federation of UDDI Registries," *Proc. First Int'l Symp. Information and Comm. Technologies,* pp. 561-566, 2003.

[24] A.I.T. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Proc. IFIP/ACM Int'l Conf. Distributed Systems Platforms (Middleware '01),* pp. 329-350, 2001.

[25] S. Sanghan and M.M. Hasan, "Intelligent P2P VoIP through Extension of Existing Protocols," *Proc. Ninth Int'l Conf. Advanced Comm. Technology (ICACT '07),* pp. 1597-1601, 2007.

[26] S. Saroiu, P.K. Gummadi, and S.D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. Ninth Multimedia Computing and Networking (MMCN '02),* 2002.

[27] B. Sapkota, D. Roman, S.R. Kruk, and D. Fensel, "Distributed Web Service Discovery Architecture," *Proc. Advanced Int'l Conf. Telecomm. and Int'l Conf. Internet and Web Applications and Services,* p. 136, 2006.

[28] C. Schmidt and M. Parashar, "A Peer-to-Peer Approach to Web Service Discovery," *World Wide Web,* vol. 7, no. 2, pp. 211-229, 2004.

[29] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM '01),* pp. 149-160, 2001.

[30] C. Williams, P. Huibonhoa, J. Holliday, A. Hospodor, and T.J.E. Schwarz, "Redundancy Management for P2P Storage," *Proc. IEEE Seventh Int'l Symp. Cluster Computing and the Grid (CCGrid '07),* pp. 15-22, 2007.

[31] L. Wu, Y. He, D. Wu, and J. Cui, "A Novel Interoperable Model of Distributed UDDI," *Proc Int'l Conf. Networking, Architecture, and Storage (NAS '08),* pp. 153-154, 2008.

[32] X. Liao, H. Jin, Y. Liu, M.N. Lionel, and D. Dafu, "AnySee: Peer-to-Peer Live Streaming," *Proc. IEEE INFOCOM,* 2006.

[33] X. Zhang, J. Liu, B. Li, and T.-S. Peter Yum, "CoolStreaming/DONet: A Data-Driven Overlay Network for Peer-to-Peer Live Media Streaming," *Proc. IEEE INFOCOM,* pp. 2102-2111, 2005.

[34] B.Y. Zhao, L. Huang, J. Stribling, A.D. Joseph, and J. Kubiatowicz, "Exploiting Routing Redundancy via Structured Peer-to-Peer Overlays," *Proc. IEEE 11th Int'l Conf. Network Protocols (ICNP '03)*, pp. 246-257, 2003.
[35] B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph, "Tapstry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," Computer Science Division of Univ. California, 2001.
[36] G. Zhou, J. Yu, R. Chen, and H. Zhang, "Scalable Web Service Discovery on P2P Overlay Network," *Proc IEEE Int'l Conf. Services Computing (SCC '07)*, pp. 122-129, 2007.

Qiang He received the first PhD degree in information and communication technology from Swinburne University of Technology (SUT), Australia, in 2009 and the second PhD degree in computer science and engineering from Huazhong University of Science and Technology (HUST), China, in 2010. He is now a research fellow at SUT. His research interests include services computing, cloud computing, P2P systems, workflow management, and agent technologies. He is a member of the IEEE.

Jun Yan received the BEng and MEng degrees in computer application technologies from Southeast University, Nanjing, China, in 1998 and 2001, respectively, and the PhD degree in information technology from Swinburne University of Technology, Melbourne, Australia, in 2004. Currently, he is a senior lecturer in the School of Information Systems and Technology, University of Wollongong, Australia. His research interests include software technologies, workflow management, service-oriented computing, and agent technologies.

Yun Yang received the BSci degree from Anhui University, Hefei, China, in 1984, the MEng degree from the University of Science and Technology of China, Hefei, in 1987, and the PhD degree from the University of Queensland, Brisbane, Australia, in 1992, all in computer science. Currently, he is a full professor in the Faculty of Information and Communication Technologies at the Swinburne University of Technology, Melbourne, Australia. Prior to joining Swinburne as an associate professor, he was a lecturer and senior lecturer at Deakin University from 1996-1999. Before that, he was a (senior) research scientist at DSTC Co-operative Research Centre for Distributed Systems Technology from 1993-1996. He also worked at Beihang University from 1987-1988. He has coauthored one book and published more than 180 papers in journals and refereed conferences. His current research interests include software technologies, cloud computing, p2p/grid/cloud workflow systems, and service-oriented computing.

Ryszard Kowalczyk received the PhD degree from the Silesian University of Technology, Poland, in 1990. He is a full professor of intelligent systems in the Faculty of Information and Communication Technologies (ICT) at Swinburne University of Technology, Melbourne, Australia. His research interests include intelligent systems, agent technology and collective intelligence, and their applications in a wide range of complex real-world problems. He has authored four patents and more than 160 articles in international journals and refereed conference proceedings. He is an editor-in-chief of *Transactions on Computational Collective Intelligence* (Springer) and has served on a number of editorial and advisory boards of international journals and scientific organizations.

Hai Jin received the PhD degree in computer engineering from the Huazhong University of Science and Technology (HUST), China, in 1994. He is a Cheung Kung Scholars chair professor of computer science and engineering and the dean of the School of Computer Science and Technology at HUST. He worked at The University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He is the chief scientist at ChinaGrid, the largest grid computing project in China, and the chief scientist of the National 973 Basic Research Program Project of Virtualization Technology of Computing System. He is the member of the Grid Forum Steering Group (GFSG). He has coauthored 15 books and published more than 400 research papers. His research interests include computer architecture, virtualization technology, cluster computing and grid computing, peer-to-peer computing, network storage, and network security. He is the steering committee chair of the International Conference on Grid and Pervasive Computing (GPC), the Asia-Pacific Services Computing Conference (APSCC), the International Conference on Frontier of Computer Science and Technology (FCST), and the Annual ChinaGrid Conference. He is a member of the steering committee of the IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid), the IFIP International Conference on Network and Parallel Computing (NPC), the International Conference on Grid and Cooperative Computing (GCC), the International Conference on Autonomic and Trusted Computing (ATC), and the International Conference on Ubiquitous Intelligence and Computing (UIC). In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. He was awarded the Excellent Youth Award from the National Science Foundation of China in 2001. He is a senior member of the IEEE and ACM.