

Chord4S: A P2P-based Decentralised Service Discovery Approach

Qiang He^{1,3}, Jun Yan², Yun Yang¹, Ryszard Kowalczyk¹, Hai Jin³

¹ Faculty of Information and Communication Technologies
Swinburne University of Technology, Australia 3122
qhe@ict.swin.edu.au {yyang, rkowalczyk}@swin.edu.au

² School of Information Systems and Technology
University of Wollongong, Australia 2522
jyan@uow.edu.au

³ School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, China 430074
hjin@hust.edu.cn

Abstract

Service-oriented computing is emerging as a paradigm for developing distributed applications. A critical issue of utilising service-oriented computing is to have a scalable, reliable and robust service discovery mechanism. However, traditional service discovery methods using centralised registries can easily suffer from problems such as performance bottleneck and vulnerability to failures in the large scalable service network, thus functioning abnormally. To address these problems, this paper proposes a peer-to-peer based decentralised service discovery approach named Chord4S. Chord4S utilises the data distribution and lookup capabilities of the popular Chord to distribute and discover services in a decentralised manner. Data availability is further improved by distributing service descriptions of functionally-equivalent services to different successor nodes that are organised into a virtual segment in the Chord circle. In addition, the Chord routing protocol is extended to support efficient discovery of multiple services with single request. This enables late negotiation of service level agreements between a service consumer and multiple service providers. The experimental evaluation shows that Chord4S achieves higher data availability and provides efficient query with reasonable overhead.

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). However, centralised

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 the open environment. This inherent disadvantage prevents Web services from being applied in large scalable service networks. As a service environment is largely distributed, decentralised approach appears to be the most natural way to support service discovery.

The peer-to-peer (P2P) technology provides a universal approach to improve flexibility and scalability of distributed systems by removing centralised infrastructures. In areas such as file sharing, Voice over Internet Protocol (VoIP) and video streaming, 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 [18], CAN [14], Pastry [16] and Tapstry [21], have some characteristics that are suitable for facilitating efficient decentralised 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 decentralised service discovery approach consists of a set of distributed nodes which 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 request can be submitted to any node and this node, if does not hold the required data elements, is able to route the request to appropriate nodes for resolution. Matching services are retrieved and returned to the service consumer as results of the request.

Although structured P2P can potentially improve the scalability of service discovery, directly applying

DHT-based P2P approaches to decentralised service discovery may be weak in guaranteeing the availability of published data elements. 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 is not able to discover any of these services. This disadvantage may result in serious problems in open and dynamic service environments where unexpected failure of node cannot be avoided.

This paper proposes Chord4S, a Chord-based decentralised service discovery approach. Chord is selected because it is well recognised for its flexibility and scalability and is considered suitable in large-scale service environments. Chord4S takes advantages of the basic principles of Chord for nodes organisation, data distribution and query routing to improve the efficiency of service discovery. Chord4S also largely improves data availability by distributing functionally equivalent services to different successor nodes purposefully. In case one node fails, a service consumer is still able to find functionally equivalent services from other successor nodes. Furthermore, Chord4S extends Chord's original routing protocol to support discovery of multiple functionally-equivalent services at different nodes with one request, which is necessary for negotiation of service level agreement (SLA) and selection of optimal service providers [3].

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

2. Related work

2.1 Centralised service discovery

Centralised client/server model has been adopted for service discovery since SOC emerged. UDDI [7] has been recognised as the most popular discovery mechanism for Web services. At present, several software vendors include UDDI support as a key feature of their software products to provide comprehensive solution for application and service integration challenges. Those softwares include 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, centralised infrastructures inherently suffer from poor performance in complicated open environments that demand high scalability.

2.2 Decentralised service discovery

Decentralised service discovery is considered as a promising approach to address the problems caused by centralised infrastructures. In particular, some preliminary research has been conducted to utilise P2P computing for service discovery. To name a few, [17] describes a system that implements an Internet-scale DHT. The system supports searches using keywords, partial keywords and wildcards. To preserve the locality while mapping data elements to the index space, the system uses recursive, self-similar Space Filling Curves (SFC). [22] presents 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 load on peer nodes, a multi-layer P2P overlay network is constructed to aggregate similar indexing keys. [8] presents 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. [12] presents 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 complicate query. Web Services Dynamic Discovery (WS-Discovery) [4], 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. In [11], authors propose that service providers themselves should take the responsibility to maintain their own service descriptions in a decentralised environment. Based on this concept, a decentralised 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.

The research reported in this paper is similar to the work presented in [11] in using layered service identifiers to control the distribution of service descriptions. This research further improves the data availability in open environment and supports efficient query of a customised number of required service providers, as detailed in Sections 3 and 4.

3. Service description distribution

3.1 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, one of the general consistent hashing functions, as its hashing function to generate identifiers for the data and nodes. Chord organises 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:

$$\begin{aligned} \text{identifier}_{\text{node}} &= \text{hash}_{\text{SHA-1}}(\text{IP}_{\text{node}}) \\ \text{identifier}_{\text{data}} &= \text{hash}_{\text{SHA-1}}(\text{Description}_{\text{data}}) \end{aligned}$$

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

$$\begin{aligned} \text{put}(\text{identifier}_{\text{data}}, \text{data/pointer to data}) \\ \text{lookup}(\text{identifier}_{\text{data}}) \end{aligned}$$

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 decentralised service discovery, information about available services, i.e., service description, needs to be distributed at different nodes. However, DHT is focused 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 service-oriented 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.AVI.Decoder”. 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.AVI.Decoder”.

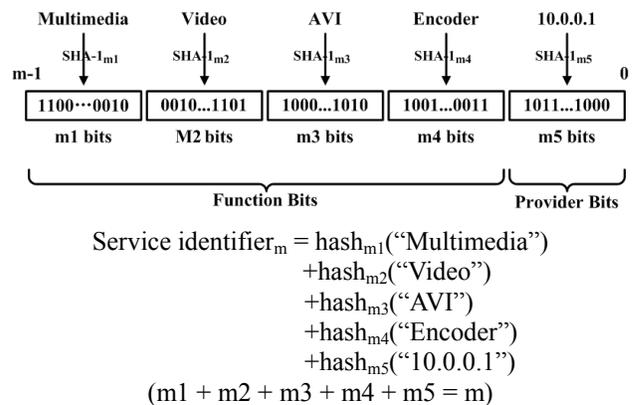
3.2 Service description distribution in Chord4S

There are two traditional approaches to address the data availability issue discussed in Section 3.1, replication (i.e., storage of multiple copies of a service description at different nodes) [10] and redundancy (i.e., storage of redundant information along with the service description) [19, 20]. 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.

Description and categorisation 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 [5] and WSMML – Web Service Modeling Language [15]. 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” (6-layered) and “Multimedia.Video.AVI.Encoder” (4-layered). The number of layers is 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 utilised 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 service descriptions and the rest for provider bits. A sample of service identifier consisting of five layers is presented in Figure 1. The function bits are used for the functional service match-making in service discovery. And the main function of the provider bits is to distinguish and distribute functionally equivalent services. Using SHA-1, the probability of hashing two service descriptions to a same value is negligible as long as the length of provider bits is large enough.



Note: Here “+” executes connection of bits.

Figure 1. Service identifier generated from hierarchical service description

Identifiers generated from functionally equivalent services differ from each other in a certain number of 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 organised into a circle in ascending order. Therefore, functionally equivalent service descriptions 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 can allow for service providers to publish service descriptions in mixed structures. For example, in an application with a maximum of 5-layered 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 using hash_{m_1} (“Multimedia”), hash_{m_2} (“Video”) and hash_{m_3} (“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 Figure 2 to illustrate the specific situation.

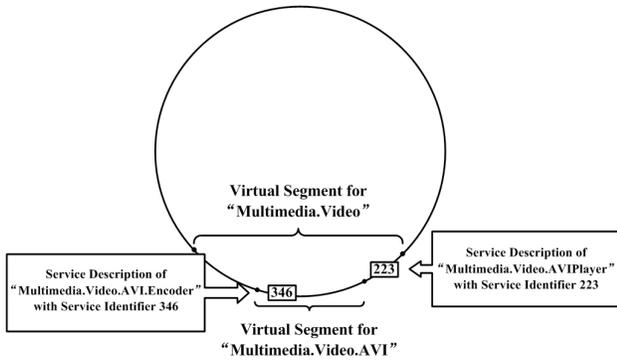


Figure 2. Virtual segment for “Multimedia.Video” and “Multimedia.Video.AVI”

Besides distinguishing functionally equivalent services, the provider bits can be used for other purposes. The behaviour 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, can be taken into consideration when looking up service providers in complicated applications. For example, BPEL and OWL-S descriptions can be converted into finite automaton through several methods [6, 9, 13]. Then the results from hashing the finite automaton or the path finite automaton (PFA) generated from the finite automaton can

be put in the provider bits of the service identifiers to enable semantic-enhanced service discovery [8]. Furthermore, provider bits can also accommodate reputation information to facilitate reputation-enhanced service discovery [23].

3.3 Discussion

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 network overlay 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:

$$2^x \geq k - 1 \quad (1)$$

Hence,

$$x \geq \log_2(k - 1) \quad (2)$$

Hashed into the identifier space, the 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.

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

Hence,

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

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.

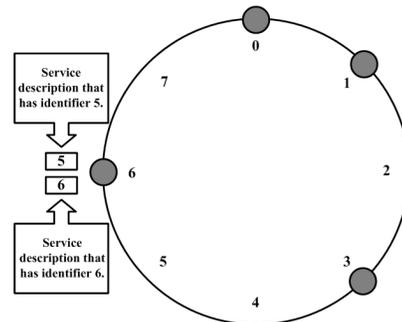


Figure 3. Uneven distribution example

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. Figure 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 Figure 3 may often occur. The effect of data distribution may be reduced. However, the more nodes joined the system, the more effective the data distribution mechanism will be. It is also the essence of all P2P-based applications.

4. Service discovery query processing

This section presents how routing of query messages is performed in Chord4S based on the data distribution mechanism described in Section 3.

4.1 Query forwarding

In Chord, a service consumer could easily get a list of matched services whose service descriptions are stored at the same successor node. However, the distribution of service descriptions described in Section 3.2 decreases 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 request, 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. However, the routing performance degradation is negligible where details are addressed in Section 5.

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 node. Each initiated query message contains the following basic information: a counter and a given service identifier. The 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 required service identifier. If the result equals to the required service identifier, then the matching succeeds. Logically, this AND operation is used to extract the ID of the virtual segment that the node belongs to and

to find out if this ID equals to the function bits in the required service identifier. A sample of matched service identifiers is shown in Figure 4.

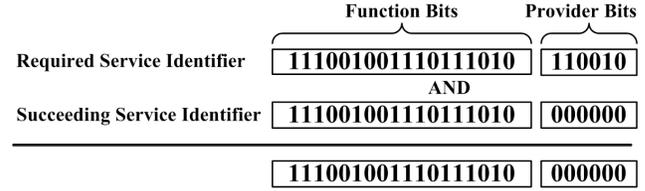


Figure 4. Service matching operation

After a matched service description is found, a query will still be passed around 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 it does, 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 3.2, 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 0’s, 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 (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 Figure 5 with predecessor of local variables and procedure calls omitted. When looking for matched successor nodes, node d 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. The function *closest_preceding_finger* is used to request n to find the node known by n that most closely precedes *message.id*. But the implementation is different from Chord as a result of our novel hierarchical structure for service descriptions.

```

//when node n receives a query message
n.find_successor(message)
  if (message.id <= id)
    if message.counter == 0
      return message;
    for i = 1 upto list_of_service_ids.length
      if match_making(message.id, list_of_service_ids[i])
        message.counter = message.counter - 1;
        message.list_of_candidate_service_providers.add(list_of_service_descriptions[i]);
        if message.counter == 0 break;
      n' = find_next_successor(message.id);
      if n' != null
        return n'.find_successor(message);
    else return message;
  if (message.counter != 0)
    n' = closest_preceding_finger(message.id);
    return n'.find_successor(message);

//return next closest successor node
n.find_next_successor(id)
  max_id = get_max_potential_node_id;
  min_id = get_min_potential_node_id;
  for i = 1 upto m
    if (finger[i].node C (min_id, max_id))
      return finger[i].node;
  return n;

//return closest finger preceding id
n.closest_preceding_finger(id)
  for i = m downto 1
    if (finger[i].node C (n, id))
      return finger[i].node;
  return n;

```

Figure 5. Pseudocode for finding successor operation

4.2 Performance analysis

The service discovery process in Chord4S is different from traditional approaches based on 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, \dots, p_m be the successor nodes that succeed service description s , sorted by ascending node identifiers. Assume that functionally equivalent services are completely distributed at different successor nodes, i.e., each successor node can

only store one matched service. In general, node p_l , the one with the lowest node identifier, will be populated at the edge of the certain virtual segment that aggregates descriptions of service s . Therefore p_l 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 $O(\log N)$ steps to find the first successor node p_l . The maximum path length from p_l 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_l 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 steps needed for p_l to find the next matched successor node (if there is any) is $O(\log(N/5))$. In this case, the number of necessary forwardings will be $O(\log N) + O(\log(N/5)) = O(\log N)$.

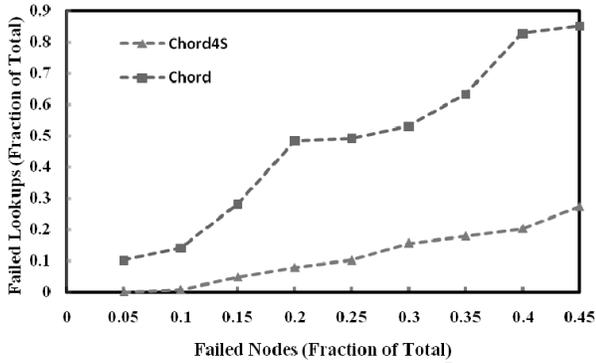
When there are several matched successor nodes in p_l 's finger table, i.e., p_2, p_3, \dots, p_m , p_l will route the query message directly to its closest successor, i.e., p_2 . Following this, p_2 may route the query message directly to p_3 if p_3 exists. The query message will be passed along the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

The above analysis follows the assumption that all service descriptions strictly conform to the structure described in Section 3.2 and the function bits are obtained by hashing complete four-layered service description.

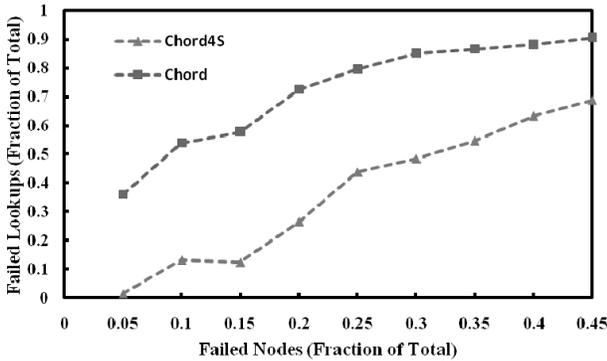
5. Experimental evaluation

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, as proven in [12], Chord4S inherits good scalability with low communication cost and state maintenance cost for service discovery. Data availability and routing performance were evaluated particularly because they are of great importance in Chord4S. Simulations were performed in overlay networks consisting 2^7 (128), 2^{10} (1024) and 2^{13} (8192) nodes, in order to access the performance of Chord4S in environments on different scales.

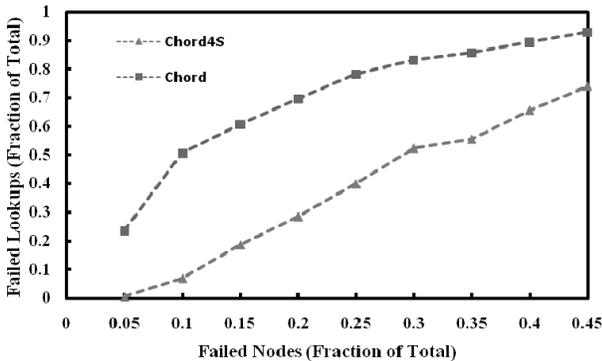
To evaluate data availability, the fraction of the failed lookups in the case of random node failure is measured. The fraction of failed nodes is increased from 5% to 45% gradually by 0.5% for each simulation. The results, as shown in Figure 6, indicate that the Chord4S curves always start with a much lower point and continue to stay at lower points compared to Chord, which means Chord4S always give better data availability than Chord.



(a) In networks consisting of 2^7 nodes



(b) In networks consisting of 2^{10} nodes



(c) In networks consisting of 2^{13} nodes

Figure 6. Data availability

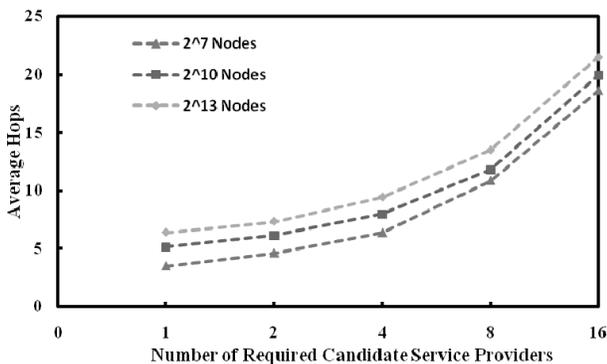


Figure 7. Routing performance

To evaluate routing performance, the average hops needed for a query of a certain number of services are measured. The number of required services per request is $k = 2^p$ with p from 0 to 4. Note that when p is assigned 0, the discovery process equals to that of Chord because when only one service is required, the request message will not be routed by the target successor node. In the experiments nodes were picked randomly to send queries of service identifiers. As shown in Figure 7, the average hops increase 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 step to find another matched service description because they are distributed next to each other. It is clear that with reasonable extra hops, multiple functionally-equivalent services can be found with data availability maintained at a higher level than Chord. This feature makes Chord4S more feasible for applications in dynamic and distributed service environments.

6. Conclusion and future work

Service discovery is a critical component of service-oriented computing. Over recent years, peer-to-peer based service discovery has attracted researchers' attention after the deficiencies of centralised service discovery are identified. This paper has proposed Chord4S, a peer-to-peer based approach for decentralised service discovery. To improve data availability, Chord4S distributes the descriptions of functionally-equivalent services. An efficient routing algorithm is provided to facilitate queries of multiple candidate service providers. Chord4S is scalable 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 addition, Chord4S will be applied to practical applications and environments, such as service composition scenarios and mobile ad hoc network applications.

Acknowledgement

This work is partly funded by the Australian Research Council Discovery Project Scheme under grant number DP0663841.

References

- [1] North American Industrial Classification Scheme (NAICS) codes, <http://www.naics.com/>.
- [2] Universal Standard Products and Services Classification (UNSPSC), <http://www.unspsc.org/>.
- [3] Ardagna, D., Comuzzi, M., Mussi, E., Pernici, B., Plebani, P.: PAWS: A Framework for Executing Adaptive Web-Service Processes. *IEEE Software*. 24(6), 39-46 (2007)
- [4] Beatty, J., Kakivaya, G., Kemp, D., Kuehnel, T., Lovering, B., Roe, B., St. John, C., Schlimmer, J., Simonet, G., Walter, D., Weast, J., Yarmosh, Y., Yendluri, P., Web Services Dynamic Discovery (WS-Discovery), <http://specs.xmlsoap.org/ws/2005/04/discovery/ws-discovery.pdf>, 2005.
- [5] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A.: OWL Web Ontology Language Reference. (2004)
- [6] Cheng, Z., Singh, M. P., Vouk, M. A.: Verifying Constraints on Web Service Compositions. *Real World Semantic Web Applications*. June (2002)
- [7] Clement, L., Hatley, A., von Riegen, C., Rogers, T., UDDI Version 3.0.2: OASIS, http://www.uddi.org/pubs/uddi_v3.htm, 2004.
- [8] Emekçi, F., Sahin, O. D., Agrawal, D., Abadi, A. E.: A Peer-to-Peer Framework for Web Service Discovery with Ranking. In: *IEEE International Conference on Web Services (ICWS'04)*, pp. 192-199. IEEE Computer Society, San Diego, California, USA, (2004)
- [9] Foster, H., Uchitel, S., Magee, J., Kramer, J.: Model-based Verification of Web Service Compositions. In: *18th IEEE International Conference on Automated Software Engineering (ASE'03)*, pp. 152-163. IEEE Computer Society, Montreal, Canada, (2003)
- [10] Gopalakrishnan, V., Silaghi, B. D., Bhattacharjee, B., Keleher, P. J.: Adaptive Replication in Peer-to-Peer Systems. In: *24th International Conference on Distributed Computing Systems (ICDCS'04)*, pp. 360-369. IEEE Computer Society, Hachioji, Tokyo, Japan, (2004)
- [11] Hu, T. H.-t. Seneviratne, A.: Autonomic Peer-to-Peer Service Directory. *IEICE Transaction on Information System* E88-D(12), (2005)
- [12] Li, Y., Zou, F., Wu, Z., Ma, F.: PWS: A Scalable Web Service Discovery Architecture Based on Peer-to-Peer Overlay Network. In: *6th Asia-Pacific Web Conference on Advanced Web Technologies and Applications (APWeb'04)*, pp. 291-300. Hangzhou, China, (2004)
- [13] Narayanan, S. McIlraith, S. A.: Simulation, Verification and Automated Composition of Web Services. In: *11th International World Wide Web Conference (WWW'02)*, pp. 77-88. ACM, Honolulu, Hawaii, USA, (2002)
- [14] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content-Addressable Network. In: *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01)*, pp. 161-172. ACM, San Diego, CA, USA, (2001)
- [15] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D., The Web Service Modeling Language (WSML) v0.21, <http://www.wsmo.org/TR/d16/d16.1/v0.21/>, 2005.
- [16] Rowstron, A. I. T. Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware'01)*, pp. 329-350. Heidelberg, Germany, (2001)
- [17] Schmidt, C. Parashar, M.: A Peer-to-Peer Approach to Web Service Discovery. *World Wide Web*. 7(2), 211-229 (2004)
- [18] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In: *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01)*, pp. 149-160. ACM Press, San Diego, California, United States, (2001)
- [19] Williams, C., Huibonhoa, P., Holliday, J., Hospodor, A., Schwarz, T. J. E.: Redundancy Management for P2P Storage. In: *7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, pp. 15-22. IEEE Computer Society, Rio de Janeiro, Brazil, (2007)
- [20] Zhao, B. Y., Huang, L., Stribling, J., Joseph, A. D., Kubiawicz, J.: Exploiting Routing Redundancy via Structured Peer-to-Peer Overlays. In: *11th IEEE International Conference on Network Protocols (ICNP'03)*, pp. 246-257. IEEE Computer Society, Atlanta, GA, USA, (2003)
- [21] Zhao, B. Y., Kubiawicz, J., Joseph, A. D.: Tapstry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing. *Computer Science Division of University California, Berkeley* (2001)
- [22] Zhou, G., Yu, J., Chen, R., Zhang, H.: Scalable Web Service Discovery on P2P Overlay Network. In: *IEEE International Conference on Services Computing (SCC'07)*, pp. 122-129. IEEE Computer Society, Salt Lake City, Utah, USA, (2007)
- [23] Zhou, R. Hwang, K.: A Robust and Scalable Reputation System for Trusted Peer-to-Peer Computing. *IEEE Transactions on Parallel and Distributed Systems*. 18(4), 460-473 (2007)