SPiDeR: P2P-Based Web Service Discovery *

Ozgur D. Sahin[1], Cagdas E. Gerede[1], Divyakant Agrawal[1],
Amr El Abbadi[1], Oscar Ibarra[1], and Jianwen Su[1]

Department of Computer Science
University of California at Santa Barbara
Santa Barbara, CA 93106
{odsahin, gerede, agrawal, amr, ibarra, su}@cs.ucsb.edu

Abstract. In this paper, we describe SPiDeR, a peer-to-peer (P2P) based framework that supports a variety of Web service discovery operations. SPiDeR organizes the service providers into a structured P2P overlay and allows them to advertise and lookup services in a completely decentralized and dynamic manner. It supports three different kinds of search operations: For advertising and locating services, service providers can use keywords extracted from service descriptions (*keyword-based search*), categories from a global ontology (*ontology-based search*), and/or paths from the service automaton (*behavior-based search*). The users can also rate the quality of the services they use. The ratings are accumulated within the system so that users can query for the quality ratings of the discovered services. Finally, we present the performance of SPiDeR in terms of routing using a simulator.

1 Introduction

The adoption and evolution of the Web services technology continue to happen in many different domains from business environments to scientific applications. This technology promises to enable dynamic integration and interaction of heterogeneous software artifacts, and thereby, to facilitate fast and efficient cooperation among the entities in cooperative environments. Lately, there has been a lot of attention drawn to this promising technology from both industry and academia and it has been supported with various emerging standards and proposals such as SOAP[1], WSDL[2], BPEL[3], and OWL-S[4]; accompanying technologies such as IBM's Web Sphere, Microsoft's .NET, and Sun's J2EE; and several research efforts (see recent conferences such as [5–8]).

Web services are "software applications identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols"[9]. The main research challenges services oriented computing poses include automated composition, discovery, invocation, monitoring, validation and verification[10]. Service discovery, in particular, refers to the problem of how to search for and locate

* This research was supported in parts by NSF grants CNF 04-23336 and IIS 02-23022.

services, the descriptions of which are usually considered lying in well-defined service repositories.

Recently, a substantial progress has been done in this area thanks to several research and industrial efforts including UDDI registries [11, 12], similarity search [13], the query languages and indexing efforts [14–16], peer-to-peer (P2P) discovery techniques [17–21], semantic web approaches and ontological matching [22, 23]. These solutions, however, are typically limited for 2 reasons:

1. They are usually **centralized** where there is a single central server (e.g., UDDI registry) that keeps track of all available services. Centralized approach has well-known limitations. It is not scalable since the server has to keep information about all services and answer all queries. It is not fault tolerant because the server is a single point of failure and if the server goes down, the whole service discovery mechanism becomes unusable.
2. They usually offer limited search capabilities. There are different techniques to increase the accuracy of service discovery including functional matching (what a service does), behavioral matching (how a service performs), semantic matching (the underlying semantics of a service) and ontological matching (how a service relates to other services). Each of these provides a different metric to measure the relevance among different services and therefore, each one is important. Many existing approaches, on the other hand, concentrate on a single one or a small subset of these techniques.

In this paper, we address above issues by introducing SPiDeR, a P2P based Web service discovery framework that supports a rich set of search operations. A subset of the participating service providers (those that have good resources) are dynamically assigned as super peers and organized into a structured P2P system. Due to its P2P based design, SPiDeR distributes the tasks of indexing available services and resolving service lookups among the participants, thus providing decentralization, scalability, dynamicity, and fault tolerance. It supports 3 different types of search operations based on keywords, global service ontology, and service behavior. It also has a reputation system component for assessing the quality of the services based on the experiences of other services. The ratings given to the services are stored in the system so that users can lookup for service quality ratings when deciding which of the discovered services to use.

The rest of the paper is organized as follows. The related work is surveyed in Section 2. Section 3 introduces SPiDeR, a P2P based distributed Web service discovery framework. Section 4 describes how the different types of discovery operations (keyword-based search, ontology-based search, and behavior-based search) are supported in the framework. The quality rating scheme that enables the ranking of discovered Web services is also explained in that section. In Section 5, dynamic peer operations are discussed in detail. Those include installing and refreshing service advertisements, performing composite lookups, and indexing at the super peers. Additionally, an evaluation of SPiDeR in terms of routing performance using a simulator is presented in Section 5.4. Finally, the last section concludes the paper and outlines the future work.

2 Related Work

P2P Systems: P2P systems are a popular paradigm for exchanging data in a decentralized manner. They distribute the data and load among the peers and thus appear as a good alternative to the centralized systems. Early P2P systems, such as Napster [24] and Gnutella [25], are mainly used for file sharing. These systems are referred to as *unstructured P2P systems* [26] since the overlay network is constructed in a random manner and the data can be anywhere in the system. As a result, search and routing in these systems tend to be inefficient. *Structured P2P systems*, on the other hand, impose a certain structure on the overlay network and control the placement of data. These systems provide desirable properties such as scalability, fault tolerance, and dynamic peer insertion and departure. For example, *Distributed Hash Tables* (*DHT*s) [27–30] partition a logical space among the peers and assign each object to a peer dynamically by hashing the object's key onto the logical space. DHTs offer efficient routing and exact key lookups, which are logarithmic or sublinear in the number of peers.

Web Service Discovery: There are several proposals to increase the accuracy and efficiency of service discovery mechanisms. [13] introduces Woogle, a similarity search technique based on clustering the services according to the information gathered from their WSDL documents. It is a centralized approach and do not allow behavioral search. In [17], the authors propose a behavioral search mechanism on a P2P architecture where the service behaviors are represented as finite state automata and the services are indexed in the P2P system with keys extracted from their service automata. It only provides behavior-based search mechanism, but not the others.

SPiDeR shares similarities with [14–16]. In [14], services are represented as finite state machines which are then transformed into a form that can be indexed for efficient matching. [15] proposes an integrated directory system and a query language. The matching and ranking of the services are done via matching and ranking functions which can be customized by the users. In [16], the services are represented as message-based guarded finite state machines and behavioral signatures are used for discovering relevant services. The behavioral signatures are represented using temporal logic statements. All three approaches mentioned above consider a centralized index structure, whereas in our approach, the index is distributed over a structured P2P system.

SPiDeR also has architectural similarities with [18–21]. [18] uses DAML-S (previous version of OWL-S) for service representation and uses Gnutella P2P protocol for service discovery. [19] proposes a federation of service registries in a decentralized fashion where federations represent service groups of similar interests. Similarly, [20] and [21] consider a P2P infrastructure. While [20] uses ontologies for publishing and querying purposes, [21] describes each Web service with a set of keywords and then map the corresponding index to a DHT using Space Filling Curves. SPiDeR differs from these proposals as it can also consider the functionality and process behavior of services during discovery and supports quality rating lookups.

In terms of use of ontologies and semantic matching, we also would like to mention [22] and [23]. In [22], the authors integrate semantics and ontological matching via domain-independent and domain-specific ontologies, and propose an indexing method, namely attribute hashing. In [23], a federated registry architecture is proposed where ontologies are to provide a domain-based classification of the registries.

3 SPiDeR Overview

SPiDeR allows distributed Web service discovery over a P2P system and supports a variety of different lookup operations (those operations will be discussed in Section 4). It organizes the participants into a super-peer based structured P2P overlay and allows them to advertise their own services as well as search for other available services.[1] Chord [28] is used as the underlying P2P overlay due to its simplicity and robustness [31], though any other DHT could have been used instead. In this section we briefly introduce the Chord system and then describe the super-peer based architecture of SPiDeR.

3.1 Chord

Chord [28] is a P2P system that implements a DHT. It uses an m-bit identifier ring, $[0, 2^m - 1]$, for routing and locating objects. Both the objects and the peers in the system are assigned m-bit keys through a uniform hash function and mapped to the identifier ring. An object is stored at the peer following it on the ring, i.e., its *successor*. Figure 1 depicts a 4-bit Chord system with 5 peers. It shows the peers that are responsible for a set of keys with different IDs. For example, key 15 is assigned to its successor P_0, which is the first peer after ID 15 on the Chord ring in clockwise direction.

Routing and Lookup: Each peer maintains a finger table for efficient routing. The finger table of a peer contains the IP addresses and Chord identifiers of $O(logN)$ other peers, i.e., its neighbors, that are at exponentially increasing distances from the peer on the identifier ring, where N is the number of peers in the system. The finger table for peer $P3$ is shown in Figure 1. Peers periodically exchange *refresh* messages with their neighbors to keep their finger tables up to date. Chord is designed for very efficient exact-key lookups. A lookup request is delivered to its destination via $O(logN)$ hops. At each hop, the request is forwarded to a peer from the finger table whose identifier most immediately precedes the destination point. In Figure 1, peer $P3$'s request for key 15 is routed through $P13$ to 15's successor $P0$, by following the finger pointers.

Peer Join and Departure: Chord is a dynamic system where peers constantly join and leave the system. When a new peer wants to join, it is assigned an identifier and it sends a join request toward this identifier through an existing peer. Thus the new peer locates its successor, from which it obtains the keys

[1] In the rest of the paper, the terms *participants* and *peers* will be used interchangeably.

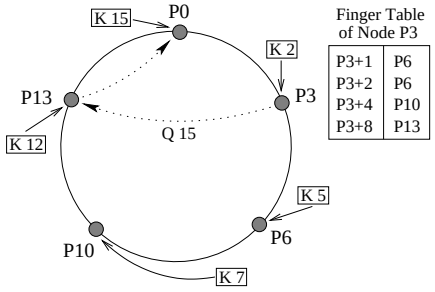

Fig. 1. 4-bit Chord System

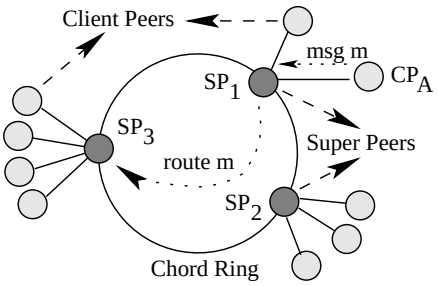

Fig. 2. Architecture of SPiDeR

it is responsible for. The affected finger tables are then updated accordingly. Similarly, upon departure of a peer, its keys are assigned to its successor and the affected finger tables are updated.

3.2 SPiDeR Architecture

Chord supports efficient exact key lookups. However dynamic peer join and departure might be costly due to finger table updates and key transfers. Additionally, Chord does not consider the heterogeneity of the peers and treats each peer equally. Thus SPiDeR uses a super peer based overlay built on top of Chord. Instead of inserting all participants into the Chord ring, only a subset of the participants are assigned as *super peers* and join the Chord ring. The super peers are selected among the peers that have good resources such as high availability, high computing capacity, etc. In this architecture, the super peers do all the indexing and query routing. The remaining peers are called the *client peers* and they use the system by connecting to a super peer. Each client peer forwards its requests to its super peer, which processes the requests on its behalf.

The super peer overlay can be maintained dynamically without any central control [32]. Each new peer joins the system as a client peer (except the initial peer starting the system). Whenever a new super peer is required (e.g., an existing super peer leaves or gets overloaded), a super peer assigns one of its client peers with good resources as a new super peer. The super peer based architecture of SPiDeR provides the following advantages:

- Peer capabilities vary widely in terms of computing/storage resources, bandwidth and availability/reliability. Peers with lots of resources will be designated as super peers and will do all the message routing and indexing. Client peers, on the other hand, just ask queries and answer service requests.
- With less peers on the Chord ring, both the routing cost and join/leave overhead are less. Client peers can join (leave) the system by simply connecting to (disconnecting from) their super peers. Thus the system is more resilient to high churn (frequent peer joins and departures).

Figure 2 depicts the architecture of SPiDeR. Note that the central UDDI registry is replaced with the Chord ring in SPiDeR. Any peer in Figure 2 (super or client) can be offering services, which are indexed within Chord by super peers. Similarly, the lookup requests are resolved in a decentralized manner by routing them to the corresponding super peer.

4 Distributed Discovery

SPiDeR organizes service providers into a structured P2P overlay that efficiently supports exact key lookups. In this section we will discuss how this overlay is used to support different types of discovery operations.

4.1 Web Service Description

Web services are software artifacts that consist of a set of operations. There are several competing and complementary languages to describe Web services such as WSDL[2], BPEL[3], OWL-S[4], WSDL-S[33], SWSL[34], and WSML[35]. Among these, WSDL defines the service interface by specifying the following:

- **Service Information:** Contains the address, name and the textual description of the service.
- **Operation Information:** Contains the name and the description of each operation.
- **Input/Output Information:** Defines the names and the types of the operation parameters.

The interface of a service describes how to access and invoke a service. WSDL-S extends WSDL by supporting inline semantic annotation. The service behavior (choreography), on the other hand, defines how to interact with the service, i.e., the possible interaction sequences the service can go through during a communication with other parties. BPEL, OWL-S, SWSL, and WSML are some examples of complimentary languages to capture service behavior.

4.2 Keyword-based Search

The first discovery method supported by SPiDeR is keyword search. In this method, each service is advertised in the system with a set of keywords. Interested parties can then locate the services they are looking for by querying the system with keywords. When keyword-based search is used, all services that are advertised for the specified keyword will be returned.

Extracting Keywords: The keywords associated with a service can be extracted from its description, e.g., its WSDL document. Popular information retrieval tools such as Smart [36] can be used to automatically extract the tokens from the description file. These tools can also be configured to remove stop words and do stemming. The tokens that appear in *name* and *description* fields can then be used as keywords. For more accuracy during keyword extraction from

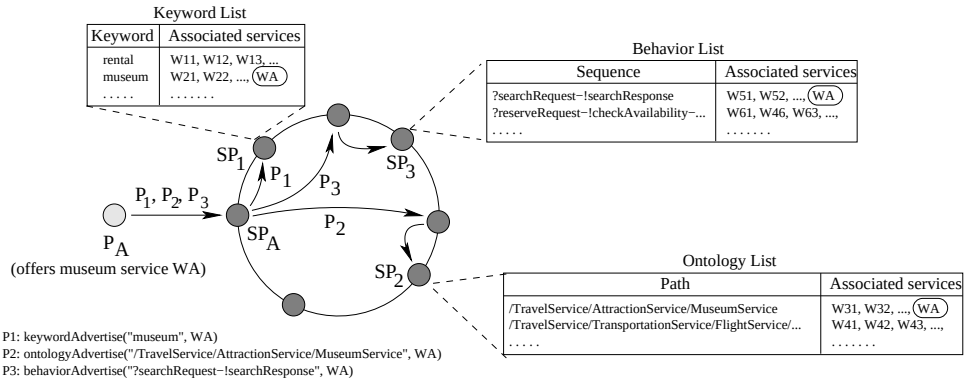

Fig. 3. Advertising Services

tokens, a thesaurus can be used or common naming conventions can be considered. A detailed discussion of extracting keywords from service description files can be found in [22].

Advertising Services: Once the keywords for advertising a service are determined, the peer offering the service sends a *keywordAdvertise* message into the system for each keyword. This message contains the keyword and the necessary information to contact the service (i.e., the address of the service). The keyword string is used as the key, so that the super peer processing the message hashes the keyword to determine its location on the Chord ring. The message is then routed and the corresponding super peer stores the service in its keyword list. For example, in Figure 3, peer P_A advertises its service W_A for keyword *museum* by sending `keywordAdvertise('`*museum*`',`W_A`)` message `P1` to its super peer SP_A. The message is then routed to SP_1, which is responsible for the key "*museum*". SP_1 stores the association $(museum, WA)$ in its keyword list.

Locating Services: When the system is queried for a keyword, the message is routed to the corresponding super peer P_S by hashing the keyword. P_S then searches its keyword list and returns all matching services to the querying peer.

4.3 Ontology-based Search

Another important search operation is category search, where the user wants to find all services within a certain category.

Common Ontology: SPiDeR assumes that there is a common domain ontology that is known by all peers in the system. This can be achieved by having each new peer download the ontology from the peer it contacts during join. This ontology identifies all possible categories for which the services can be advertised in the system. Figure 4 shows an example domain ontology for a system composed of travel related services.

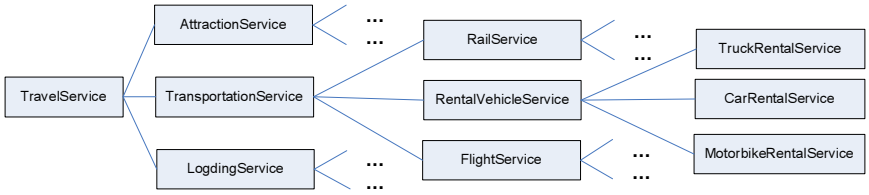

Fig. 4. Example Domain Ontology

Advertising Services: Service providers can advertise their services for each related category from the domain ontology. For each selected category, an *ontologyAdvertise* message is sent to the system. In this case, the path from the root to the corresponding node in the domain ontology is used as the key to determine the message destination. The message is routed to the super peer responsible for the path string. That peer then stores the corresponding information in its ontology list (see advertise message `P2` in Figure 3 for an example).
Locating Services: When ontology-based search is used, the path string is hashed and the corresponding super peer returns all matching services from its ontology list to the querying peer.

4.4 Behavior-based Search

Considering Web services as simple method invocations might not be sufficient in some cases. Web services can interact with other services, send and receive messages, and perform a set of activities. Such service behaviors can be defined by means of, for instance, process flow languages like BPEL. When this information is available, it can be used to improve the accuracy of service discovery by allowing users to specify the desired service behavior. For instance, Figure 5 illustrates a travel service. The automaton describes the message exchanges between the service and a user. If we examine the service behavior, we can see that the service allows its users to cancel their reservations and purchases until it finalizes the transaction. Some users may look for such specific behaviors. A behavior-based analysis can facilitate the system to differentiate among services based on their behaviors and perform the service discovery more accurately.
Advertising Services: For a given finite state automaton representing the service behavior of a Web service (such a finite automaton can be automatically extracted from the service's BPEL document[37]), we extract all the accepting paths from the automaton. An accepting path starts from the initial state of the automaton and ends in an accepting state without any loops (semantically the path implies a sequence of activities successfully performed by the service). Each accepting path is then advertised in the system using the path as a key. The entire service automaton of the service is included in the advertise message and stored in the behavior-list. For the service automaton given in Figure 5, two of the accepting paths are $<$*?searchRequest - !searchResponse*$>$ and $<$*?reserveRequest - !checkAvailability - !notAvailable*$>$. The advertise message `P3` in Figure 3 shows

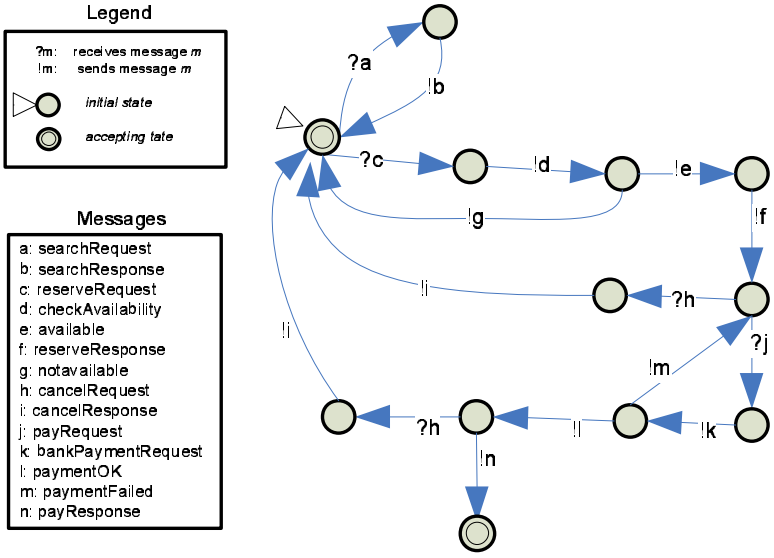

Fig. 5. Example Service Automaton

how service WA is advertised for the first accepting path above. A detailed discussion of the behavior-based search can be found in [17].

Locating Services: Users can ask behavior-based lookup queries for accepting paths they are looking for. In this case, the super peer that is responsible for that path will search through its behavior-list to find the service automata that matches with the query path and return them.

Overview

SPiDeR allows peers to search for Web services in 3 different ways: keyword-based, ontology-based, and behavior-based. All three of these techniques leverage the basic exact key lookup functionality provided by the super-peer overlay, but they incorporate different semantic meanings and thus enable SPiDeR to provide a richer set of querying capabilities. For example, a user P_x looking for services related to museums might not be able to find a relevant service S_m using keyword search just because the service is not advertised for the keyword(s) provided by P_x. However, if S_m is advertised based on ontology, P_x can locate it by issuing an ontology-based search on */TravelService/AttractionService/MuseumService*. Similarly, consider a user P_y looking for flight booking services with express delivery option. If keyword-based or ontology-based search is used, P_y will have to investigate the set of returned services to determine the ones that have express delivery option. Instead, P_y can use behavior-based search so that only the services with express delivery option are returned.

4.5 Ranking Services

In addition to the above search methods, SPiDeR provides a rating discovery mechanism. After using a Web service, a user can rate the quality of the service. The user (P_U) sends a message containing its own address, the address of the service being rated (WS), and the corresponding score (a real value between 0 and 1, where 1 is the highest score). The message is then routed in the system by hashing the address of the service being rated. The corresponding super peer stores all ratings given to WS in its rating list.

The rating of a Web service then can be retrieved by querying for its address. This query will return the average of all the scores the service had been given. The quality ratings are useful for selecting the service to use once a list of matching services are obtained.

5 Peer Operations in SPiDeR

In this section, we provide a more detailed discussion of some peer operations such as advertising services, composite lookups, and indexing services. We also show the routing performance of SPiDeR.

5.1 Advertising Services

After joining the system, each peer P periodically advertises the services it provides. For advertising a service, P_n sends the necessary information to its super peer (if it is not a super peer). This message specifies the address of the service, advertisement type(keyword-based, ontology-based or behavior-based), and the additional information (keyword, ontology path, service automaton). The super peer then routes the corresponding message within the Chord so that the responsible super peer stores the index information.

Peers periodically refresh their service advertisements to avoid stale index entries (super peers remove the index entries that are not refreshed) and to recover lost index information (if a super peer leaves without transferring its index information to another super peer).

5.2 Composite Lookups

Peers can use any of the supported search methods to locate the services they are looking for. The query message contains the address of the querying peer, query type (keyword/ontology/behavior/rating), and the corresponding arguments (keyword/ontology path/request automaton/service address, respectively). The super peer responsible for the argument receives the message and searches through the corresponding index. It then returns the list of matching services to the querying peer. SPiDeR can also be used for composite lookups such as searching for multiple keywords or for services with a keyword within a category. In this case, the user can retrieve the result of each elementary lookup and locally compute the intersection.

5.3 Indexing at Super Peers

In SPiDeR, super peers index the information about Web services they are assigned through Chord hashing. Each super peer keeps 4 different lists for supporting the corresponding discovery operations: keyword list, ontology list, behavior list, and rating list. SPiDeR is flexible in the sense that each super peer can individually choose the indexing methods it uses. For example, a peer might choose to keep each list as a sequential file, which might not be very efficient. More efficient index lookups can be achieved by using more efficient indexing schemes. For keyword, ontology, and rating lists, hashtable-like indexing methods are desirable since these lists only require exact key lookups. For behavior list, the matching regular expressions should be identified, so an RE-tree (Regular Expression tree) [38] might be suitable.

5.4 Routing Cost

We measured the performance of SPiDeR in terms of routing using a simulator implemented in Java. To measure the routing cost, we measured the average number of peers visited for routing a message in the super peer ring for different number of peers. For each message, the initiating peer is selected uniformly at random from the existing super peers and the message destination is set to a random Chord identifier. Figure 6 shows the results, where each data point is the average over 1000 runs. The routing cost is low and also increases gracefully with increasing number of peers. For example, routing a message takes 3.7 overlay hops on the average in a 50 peer system, whereas it takes 5.4 hops when the peer number increases to 500. Note that the peer numbers shown on the graph are the number of super peers and do not include the client peers. The actual number of peers in the system can be much more than the number shown in the graph. Thus we conclude that SPiDeR is both scalable and efficient in terms of routing. Compared to a centralized service discovery system, SPiDeR is more robust and scalable, and supports a richer set of discovery operations at the expense of a little routing overhead.

6 Conclusion

With the proliferation of Web services technology and the increase in the number of Web services, the service discovery problem gets more challenging. There are different dimensions like functionality, behavior and semantics each describing a service from a different perspective. In this paper, we proposed a structured P2P framework which unifies these perspectives by means of providing different search methods in a distributed environment without a central component. We believe our results will contribute to the efforts towards comprehensive service discovery systems. As future work, we plan to look into other possible perspectives that users may be interested in and improve our system with the addition of new search methods.

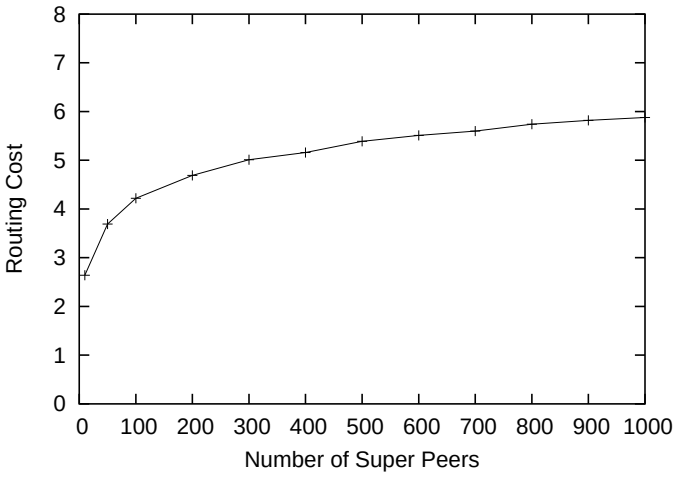

Fig. 6. Routing Performance of SPiDeR

References

1. Simple Object Access Protocol (SOAP) 1.2: http://www.w3.org/TR/SOAP/ (2003)
2. Web Services Description Language (WSDL) 2.0: http://www.w3.org/tr/ (2001)
3. Business Process Execution Language (BPEL) 2.0: http://www.oasis-open.org/committees/download.php/10347/wsbpel-specification-draft-120204.htm (2004)
4. OWL-S 1.1: http://www.daml.org/services/owl-s/1.1/ (2004)
5. Aiello, M., Aoyama, M., Curbera, F., Papazoglou, M., eds.: Proceedings of the International Conference of Service Oriented Computing (ICSOC'04), November 15-19, 2004, New York City, NY, USA,. In Aiello, M., Aoyama, M., Curbera, F., Papazoglou, M., eds.: ICSOC, ACM Press (2004)
6. Proceedings of the IEEE International Conference on Web Services (ICWS), San Diego, California, USA. (2004)
7. Proceedings of the IEEE International Conference on Services Computing (SCC), Shanghai, China. (2004)
8. Ellis, A., Hagino, T., eds.: Proceedings of the 14th international conference on World Wide Web, Chiba, Japan, May 10-14, 2005. In Ellis, A., Hagino, T., eds.: WWW, ACM (2005)
9. Web Services Architecture Requirements: http://www.w3.org/tr/wsa-reqs (2004)
10. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures and Applications. Springer (2004)
11. Binding Point: http://www.bindingpoint.com/ (2005)
12. Web Service List: http://www.webservicelist.com/ (2005)
13. Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang, J.: Similarity search for web services. In: VLDB. (2004)
14. Mahleko, B., Wombacher, A., Frankhauser, P.: A grammar-based index for matching business processes. In: ICWS. (2005)

15. Constantinescu, I., Binder, W., Faltings, B.: Flexible and efficient matchmaking and ranking in service directories. In: ICWS. (2005)
16. Shen, Z., Su, J.: Web service discovery based on behavior signatures. In: Proceedings of International Conference on Services Computing. (2005)
17. Emekci, F., Sahin, O.D., Agrawal, D., El Abbadi, A.: A peer-to-peer framework for web service discovery with ranking. In: ICWS. (2004) 192–199
18. Paolucci, M., Sycara, K., Nishimura, T., Srinivasan, N.: Using daml-s for p2p discovery. In: ICWS. (2003) 203–207
19. Papazoglou, M.P., Kramer, B., Yang, J.: Leveraging web-services and peer-to-peer networks. In: CAISE. (2003) 485–501
20. Schlosser, M., Sintek, M., Decker, S., Nejdl, W.: A scalable and ontology-based p2p infrastructure for semantic web services. In: P2P. (2002) 104–111
21. Schmidt, C., Parashar, M.: A peer-to-peer approach to web service discovery. In: WWW. (2004) 211–229
22. Syeda-Mahmood, T., Shah, G., Akkiraju, R., Ivan, A., Goodwin, R.: Searching service repositories by combining semantic and ontological matching. In: ICWS. (2005)
23. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J.: Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services. Inf. Tech. and Management **6** (2005) 17–39
24. Napster: (http://www.napster.com/)
25. Gnutella: (http://www.gnutella.com/)
26. Lv, Q., Ratnasamy, S., Shenker, S.: Can heterogeneity make gnutella scalable? In: IPTPS. (2002) 94–103
27. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: SIGCOMM. (2001) 161–172
28. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM. (2001) 149–160
29. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Middleware. (2001)
30. Zhao, Y.B., Kubiatowicz, J., Joseph, A.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley (2001)
31. Gummadi, P.K., Gummadi, R., Gribble, S.D., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of dht routing geometry on resilience and proximity. In: SIGCOMM. (2003) 381–394
32. Yang, B., Garcia-Molina, H.: Designing a super-peer network. In: ICDE. (2003) 49–60
33. Web Service Semantics - WSDL-S: http://www.w3.org/2005/04/fsws/submissions/17/wsdl-s.htm (2005)
34. Semantic Web Services Language - SWSL: http://www.daml.org/services/swsl/ (2005)
35. Web Service Modeling Language - WSML: http://www.wsmo.org/wsml/ (2005)
36. Buckley, C.: Implementation of the SMART information retrieval system. Technical Report 85-686, Cornell University (1985)
37. Fu, X., Bultan, T., Su, J.: Wsat: A tool for formal analysis of web services. In: International Conference on Computer Aided Verification. (2004)
38. Chan, C.Y., Garofalakis, M.N., Rastogi, R.: Re-tree: An efficient index structure for regular expressions. In: VLDB. (2002) 263–274