Build a Distributed Repository for Web Service Discovery Based on Peer-to-Peer Network[1]

Yin Li[2], Futai Zou[2], Fanyuan Ma[2], Minglu Li[2]

[2]The Department of Computer Science and Engineering,
Shanghai Jiaotong University, Shanghai, China, 200030
{liyin, zoufutai, ma-fy, li-ml}@cs.sjtu.edu.cn

Abstract: While Web Services already provide distributed operation execution, the registration and discovery with UDDI is still based on a centralized repository. In this paper we propose a distributed XML repository, based on a Peer-to-Peer infrastructure called pXRepository for Web Service discovery. In pXRepository, the service descriptions are managed in a completely decentralized way. Moreover, since the basic Peer-to-Peer routing algorithm cannot be applied directly in the service discovery process, we extend the basic Peer-to-Peer routing algorithm with XML support, which enables pXRepository to support XPath-based composite queries. Experimental results show that pXRepository has good robustness and scalability.

1. Introduction

Web services are much more loosely coupled than traditional distributed applications. Current Web Service discovery employs a centralized repository such as UDDI[1], which leads to a single point of failure and performance bottleneck. The repository is critical to the ultimate utility of the Web Services and must support scalable, flexible and robust discovery mechanisms. Since Web services are widely deployed on a huge amount of machines across the Internet, it is highly demanded to manage these Web Services based on a decentralized repository.

Peer-to-peer, as a complete distributed computing model, could supply a good scheme to build the decentralized repository for the Web Service discovery. Existing Peer-to-Peer systems such as CFS[3] and PAST[4] seek to take advantage of the rapid growth of resources to provide inexpensive, highly available storage without centralized servers. However, because Web Services utilize XML-based open standard, such as WSDL for service definition and SOAP for service invocation, directly importing these systems by treating XML documents as common files will make Web Service discovery inefficient. INS/Twine[5] seems to provide a good solution for building the Peer-to-Peer XML repository. However, INS/Twine does not provide a solution to provide XPath-like query.

[1] This paper is supported by 973 project (No.2002CB312002)of China, and grand project of the Science and Technology Commission of Shanghai Municipality (No. 03dz15027 and No. 03dz15028).

We designed a decentralize XML repository for Web service discovery based on structured Peer-to-Peer network named pXRepository (Peer-to-Peer XML Repository). Unlike Twine, as We allow index keys to be tree-structured or non-prefix sub-keys. For improved scalability, index entries are further organized hierarchically. We have extended the Peer-to-Peer routing algorithm based on Chord[2] for supporting XPath composite query in pXRepository. We name this algorithm eXChord (extended XML based Chord). Experimental results have shown that pXRepository has good scalability and robustness.

2. System Overview

pXRepository is a Peer-to-Peer XML storage facility. Each peer in pXRepository acts as a service peer (SP for simplicity), which not only provides Web service access, but also acts as a peer in the Peer-to-Peer XML storage overlay network. The architecture of the service peer in pXRepository is shown in Fig. 1.

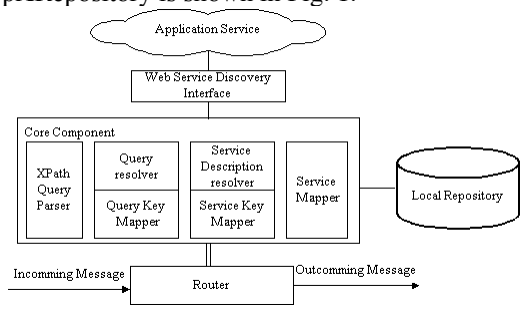


Fig. 1. The architecture of the service peer

In pXRepository, XPath[6] is used as query language for retrieving XML documents stored over the Peer-to-Peer storage network. Each SP consists of three active components called the Web Service Discovery Interface, the core component and the router, and a passive component called the local repository. Web Service Discovery Interface provides access interface to publish or locate Web services and also exposes itself as a Web service. The service description resolver is responsible for extracting key nodes from a description. Each key node extracted from the description is independently passed to the service key mapper component, together with service description or query. The service key mapper is responsible for associating HID(Hash ID) with each key node. It does this by hashing the key node. More details are given in section 3. The Query Resolver and the Query Key Mapper work almost the same way as the Service Description Resolver and the Service Key Mapper do except that the Query Resolver generates query string based on the parsing tree, which is produced by XPath parser. Service mapper is responsible for mapping HIDs to service descriptions and will return the results to the application services through the Web service discovery interface. Local repository keeps the Web service interface, service descriptions and HIDs that SP is responsible for. The router routes query requests and returns routing results.

In pXRepository, we organize every service peer in a structured Peer-to-Peer overlay network. Because Chord has the features of simplicity, provable correctness, and provable performance compared with other lookup protocols, we use Chord protocol to organize the SP's routing table.

3. Web Service Discovery in pXRepository

Service locating algorithm specifies how to route the query to the service peer who satisfies the service request. In pXRepository, the service request is expressed in XPath. However, the routing algorithm, Chord, in underlying Peer-to-Peer overlay network only supports exact-match. We extend the Chord algorithm to support XPath based match. The extended Chord algorithm is called eXChord.

In pXRepository, WSDL is used to describe the Web service interface, and the service description metadata is generated based on the content of WSDL document and the description that the user inputs before publishing. An example of Web service description metadata is shown in Fig.2.

```
 <services><service>
  <name>ListPriceService</name>
  <documentation>List the product price</documentation>
  <location>http://services.companya.com/product/
ListProductService.wsdl</location></service>
 <service><name>OrderService</name>
  <documentation>Make order to product</documentation>
  <location>http://services.companya.com/product/
OrderService.wsdl</location></service></services>
  <description><company>CompanyA</company>
 <industry>Manufactory</industry><region>China</region>
 <keyword>Automobile Price Order</keyword>
 <comments>……</comments>
 </description>
```

Fig. 2. An example of Web service description metadata in pXRepository

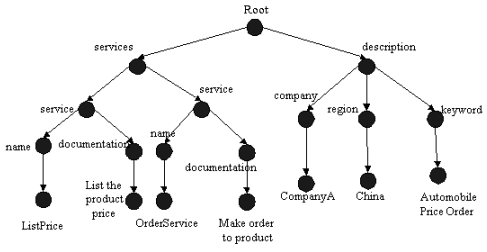
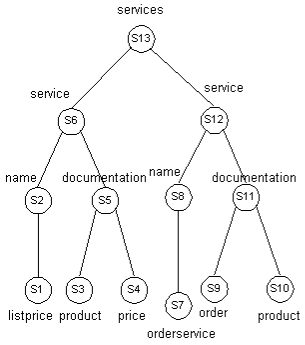


Fig. 3. A sample NVTree converted from service description shown in Fig. 2.

To publish the Web Services, the Web Service description metadata will be first converted to a canonical form: a node-value tree (NVTree). Fig.3 shows an example of the NVTree converted from the Web Service description shown in Fig. 2.

pXRepository extracts each node from the NVTree. Fig.4 shows the concatenating strings produced from the left sub-tree in Fig.3. Each node in Fig.4 is associated with a string called node key (denoted by S1,S2,…). The node key is concatenated by the child node keys and its self's node value with a slash(/) between them. If the node has multiple child nodes, each child node key is enclosed by a bracket and concatenated in left-to-right order. The concatenating process is a recursive step in post tree scanning order. The right sub-tree in Fig.3 is produced in the same way.



S1	listprice
S2	listprice/name
S3	product
S4	price
S5	[product][price]/documentation
S6	[listprice/name][[product][price]/documentation]/service
S7	orderservice
S8	orderservice/name
S9	order
S10	product
S11	[order][product]/documentation
S12	[orderservice/name][[order][product]]/documentation
S13	[S6][S12]/services

Fig. 4. Splitting a NVTree into service description key strings

Each node key is passed to the hash function to produce a HID, which will be used as a key to insert into the underlying Peer-to-Peer overlay network. In pXRepository, the hierarchical relationship of the nodes in NVTree will be preserved as shown in Fig.5. Each element in Fig.5 resides in a specific service peer in pXRepository correspond to the hash value of its node key (denoted as h(S1), h(S2),…).

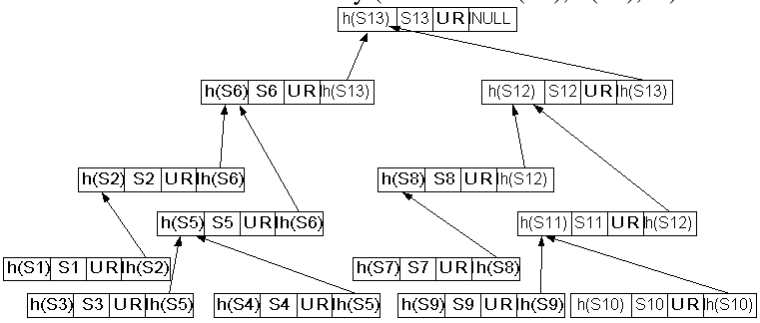


Fig. 5. Distributing node keys across pXRepository

Before presenting pXRepository service publishing and locating algorithm which is named as eXChord, we first introduce some definitions:

Definition 1. Let SD stands for a Web Service description document, then Ü(SD) stands for the URI of the document, Γ(SD) represents the NVTree of SD, and N(SD) stands for the set of NVTree nodes, where N(SD)={$N_1, N_2, \ldots, N_m$}.

Definition 2. Let N stands for a NVTree node, then P(N) stands for its parent node and K(N) represents the node key.

The pseudocode of eXChord service description publishing algorithm is given in Fig.6. Function Publish is run on peer n, take a service description (SD) as input and publishes the SD into the Peer-to-Peer overlay network.

```
1   n.Publish(SD){
2    key=hash(Ü(SD)); n`=n.Route(key); // Chord routing algorittm
3    n`.Insert(key, SD); Compute N (SD)={N1, N2 ,…,Nm };
4    for each Ni in N (SD){
5     nodekey= K (Ni);  parentkey= K ( P (Ni));
6     n.Distribute(nodekey, parentkey,SD); }
7   }
8   n.Distribute(nk, pk, SD){
9    id = hash(nk); n`=n.Route(id);
10   n`.Insert(id, nk, Ü(SD), hash(pk));
11  }
```

Fig. 6. The pseudocode of eXChord service description publishing algorithm

```
1   Let R= Φ  // R is the result set of the query
2   n.Locate(QD){
3    Compute N (QD)={N1, N2 ,…,Nm };
4    for each Ni in N (QD){
5     key=hash( K (Ni));  n`=n.Route(key);
6     NV= n`.get(key); /*NV represents the set of nodes having the
                node key value of  K (Ni) */
7      for each NVi in NV {
8       SD= n`.Match(NVi, QD); /*Macth is a recursive process finding
                    matching document set*/
9      if SD !=NULL then      R=R ∪ SD;}
10   }
11  }
12  n.Match(N, QD){
13   status = IsMatch(N, QD );/*status can be Yes, No, or Not Sure*/
14   if status=Yes then
15    return SD; // SD is the service description document that matches QD
16   elseif status=No then   return NULL;
17   else{
18    key=hash( K ( P (N))); // get parent node key hash value
19    n`=n.Route(key);   NV= n`.get(key);
20    for each NVi in NV{
21     SD= n`.Match(NVi, QD);
22     if SD !=NULL then   R=R ∪ SD;}
23   }
24  }
```

Fig. 7. The pseudocode of eXChord service locating algorithm.

To search for a Web Service, the client must specify the query requirement, which is expressed in XPath language. pXRepository supports composite XPath queries, and each XPath query only contains text matching constraints. An XPath query can be converted to a tree called XPTree. Because each node key in NVTree preserves the sub-tree structure information, the Web Service searching process can be a sub-tree matching problem.

To present the Web Service locating algorithm, we first introduce some definitions:

Definition 3. Let QD stands for a XPath query, then Γ(QD) represents the XPTree of QD, and $\aleph$(QD) stands for the set of leaf nodes, where $\aleph$(QD)={N_1, N_2 ,…,N_m }.

Definition 4. Let N stands for a XPTree node, then κ(N) represents the value of the node.

The pseudocode of eXChord service locating algorithm is given in Fig.7. Function Locate is run on peer n, take query requirement QD as its input and searches the Peer-to-Peer overlay network for the services that satisfy its requirements.

4. Evaluation and experimental results

In this section, we evaluate pXRepository by simulation and compare pXRepository with centralized service management approach such as UDDI. We compared latency, space overhead, load, and robustness of pXRepository with UDDI.

We use the Georgia Tech Internetwork Topological Models (GT-ITM)[7] to generate the network topologies used in our simulations. We use the "transit-stub" model to obtain topologies that more closely resemble the Internet hierarchy than pure random graph. An Internetwork with 600 routers and 28800 service peers (node for simplicity) are used in our experiment.

4.1 Latency

We evaluate the latency metric in the number of the hops in the network. Fig.8 shows the effect of number of nodes on latency. Since the routing table of pXRepository is same as that of Chord, which has the logarithmic relationship between logical hops and number of the nodes. If the average latency of single logical hop is κ, thus, latency of pXRepository $Latency_{\text{pXRepository}} = \kappa \times \log(N_H)$.

Experimental results further show that the latency of UDDI is roughly 80. This is because the logical hops of UDDI is 2, and has nothing to do with N_H. Although latency of pXRepository is higher than that of UDDI, pXRepository has lower space overhead, lower load, and good robustness (refers to 4.2, 4.3, 4.4).

4.2 Space overhead

In order to analyze conveniently, we first give a definition and two assumptions:

Definition 5. The memory size of routing ID and corresponding IP is called memory unit, size of which is σ.

Assumption 1. Node ID and Web service description ID are distributed uniformly in ID space, and there are n service peers in the system, each peer publishes m service description documents in average. For each service description, s keys will be generated by extracting the concatenating strings from the service description.

Assumption 2. The average overhead of each service description item is k× σ.

Overhead of pXRepository is:

$$Space_{\text{pXRepository}} = \log(n) \times \sigma + m \times s \times k \times \sigma \quad (1)$$

Since UDDI maintains all information in central repository, overhead of UDDI is:

$$Space_{UDDI} = n \times m \times k \times \sigma \quad (2)$$

For $m = 80$, $k = 2$, $s=10$ and $\sigma = 200$, the effect of number of nodes N_H on space overhead is shown in Fig.9. Experimental results in Fig.9 reveal that the space overhead of pXRepository is much better than that of UDDI.

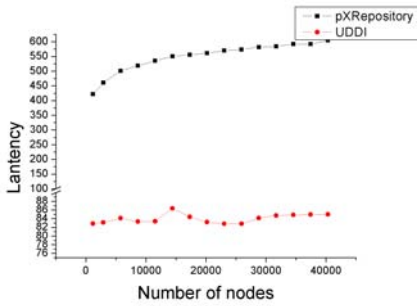


Fig. 8. Effect of number of nodes on latency

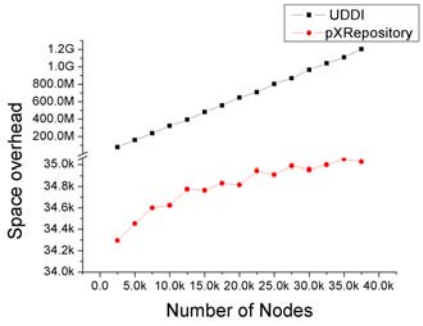


Fig. 9. Effect of number of nodes on Space overhead

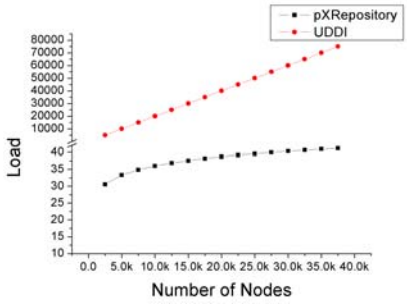


Fig. 10. Load of pXRepository and UDDI

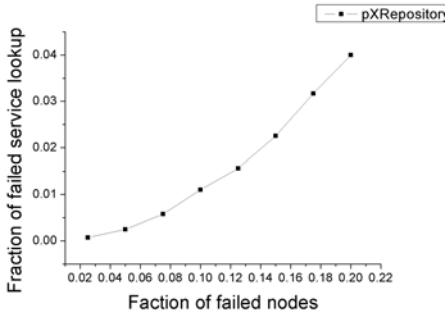


Fig. 11. The effect of node failure

4.3 Load

Load is an important metric to evaluate Web service management approach. This paper uses the number of messages in and out of the node as a metric to evaluate the load. We assume that each service lookup query will generate 3 keys in average, and each key will be used to locate the service. Fig.10 shows the comparisons in loads between pXRepository and UDDI. The load of UDDI increases linearly with the number of nodes, however, the load of pXRepository and the number of nodes is in logarithmic relationship.

4.4 Robustness

In this experiment, we consider 28800 nodes with each node distribute 10 pieces of service descriptions, and each service description will generate 10 keys. We also assume that each service query requirement issued by client user will generate 2 keys covered by the service description keys. Then we randomly select a fraction of the nodes that fail. After the failures occur, we wait for the network to stabilizing, and then measure the fraction of the keys that could not be located correctly. Fig.11 plots the effect of node failure on service lookup.

Since in eXChord algorithm used in pXRepository, for each service key, there will be a service description copy distributed in the underlying Peer-to-Peer overlay network, the client can still continue to locate the appropriate service description by using other keys in case some service peers fail.

5. Conclusions

In this paper, we propose a distributed XML repository, based on a Peer-to-Peer infrastructure called pXRepository for Web Service discovery. In pXRepository, the Web service descriptions are managed in a totally distributed way, which avoids single point of failure and can be scalable and robust. We also extend the basic Peer-to-Peer routing algorithm with XPath support.

References:

1. http://www.uddi.org/, UDDI Version 3.0, Published Specification.
2. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: a scalable Peer-to-Peer lookup service for Internet applications. Proceedings of ACM SIGCOMM`01, San Diego, September 2001.
3. F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. SOSP ’01, October 2001.
4. P. Druschel and A. Rowstron. PAST: A large-scale persistent Peer-to-Peer storage utility. In Proc. HOTOS Conf., 2001.
5. M. Balazinska, H. Balakrishnan, and D. Karger, “INS/Twine: A scalable Peer-to-Peer architecture for intentional resource discovery,” in Proceedings of the International Conference on Pervasive Computing, August 2002.
6. W3C. XML Path Language (XPath) 1.0. http://www.w3.org/TR/xpath, November 1999.
7. Zegura, E. w., Calvert. k., and Bhattacharjee, S. How to model an Internetwork. In Proceed-ings of IEEE INFOCOM (1996).