Web Services Publishing and Discovery on Peer-to-Peer Overlay

JianJun Yu  Rui Chen  Hao Su  Shengmin Guo
Department of Computer Science
Beihang University, Beijing, 100083
{yujj, chenrui, suhao, guosm}@nlsde.buaa.edu.cn

Abstract

Centralized UDDI currently becomes harder to catch up with the need of Internet-scale service publishing and discovery. Peer-to-Peer technology with its advantages of scalability, fault-tolerance and dynamics is suitable for loose-coupled Web services’ distribution in nature and prevailing nowadays. In this paper, we present a distributed, collaborative Peer-to-Peer service overlay suitable for service publishing and discovery based on Skip Graph which could be organized by planar model or tree model according to the scale of services. We propose a prefix-ordered clustering mechanism that congregates the keys within a fixed prefix into the same bucket and make the smallest one appeared as the key in the service overlay which would support prefix, range query and semantic matching on Web services. We then analyze the performance of our proposed with simulation that validates the effectiveness of the service overlay.

1. Introduction

Web services have been well acknowledged as distributed and loose-coupled components to realize sharing and interchanging enterprise resources on web. These services are universally distributed and of various kinds whilst with quite a diverse description. Therefore, how to integrate and find the exact service from massive services to meet e-business demand has become the current focus of research on service discovery.

Currently, people always publish service descriptions to UDDI (Universal Description, Discovery and Integration) registries as described at SOA (Service-Oriented Architecture). Nevertheless, this centralized mechanism meets insurmountable problems which embarrasses its massive popularization such as one point of failures, the poor maintenance mechanism of updated service information and low efficiency of service finding. In recent years, distributed mechanism for service discovery is advanced, especially P2P overlay network which brings in scalability, fault-tolerance, and self-organizing nature into Web services world and helps to eliminate shortcomings of current UDDI Web Services discovery mechanism.

In this paper, we would like to apply P2P overlay network to construct service registry rather than centralized UDDI. In addition, we use currently prevalent distributed data structure Skip Graph [1] [7] to construct service overlay network. Skip Graph is a randomized, balanced-tree data structure by adding redundant connectivity and multiple handles for locality-preserving distribution of tree structures which keeps all keys appeared in sorted order. We adopt Skip Graph for two reasons. First, it doesn’t use DHT (Distributed Hash Table) as indexing and routing algorithm which would lose the semantics of Web services descriptions and only support searching in hashed key space. Second, it supports range query and complex search which play important roles in Web services discovery.

The main contribution of this paper is constructing a service overlay as service registry supporting prefix search, range query and thus supporting service publishing, discovery and semantic matching. This service overlay can be organized as planar model or tree model to accommodate different application requirements considering the scale of Web services. In addition, we use prefix-ordered mechanism to cluster similar keys with same common longest prefix into the same bucket which could support better prefix search.

The rest of this paper is organized as follows. The second part gives the related work. The third part gives the mechanism how to split the service descriptions suitable for service publishing. The fourth part introduces the planar and tree model on Skip Graph for service discovery. The fifth part presents the enhanced algorithm of P2P overlay network supporting prefix search and range query. The sixth part presents the results of our simulation study. The final part is conclusion and future work.
2. Related Work

Service publishing and discovery is closely related to three service searching problems: text document matching, UDDI registry and P2P Web services query.

**Text document matching:** A lot of solutions for document matching and classification have been provided in information retrieval field. Web service description documents (such as WSDL) are essentially text documents, so these solutions can also be applied to service discovery. For example, we can acquire about 27,100 WSDL (Web Service Description Language) result records from Google. However, these approaches are insufficient for Web services because they ignore structure information which would include the underlying semantics of services. In [5], the authors provided a Web service search engine named Woogle which considered the structure and semantics of services. However, its centralized discovery mechanism could not solve the problems like scalability and one point of failures. In addition, it could not work well as service registry and was hard to publish services.

**UDDI Registry:** UDDI with its centralized mechanism also meets the scalability problem which embarrasses its prevalence. METEOR-S WSDI [14] provided an ontology based multiple UDDI registries which organized the registries into domains and enabled domain based classification of Web services. This architecture helped to cluster semantic similar services into same registry and made range query easier. However, it’s not a pure P2P solution for service publishing and discovery which may not support self-organizing and load balancing well.

**P2P Web services query:** Current P2P Web services query solutions [9] [4] [2] [11] [6] [8] [10] [12] emerge Web Services, Semantic Web Services and Peer-to-Peer computing technologies together to provide favorable service publishing and discovery. Among these, some methods (eg. [2] [6]) didn’t focus on the P2P overlay architecture. In [9], the authors schemed unstructured P2P network Gnutella as service overlay which may not find all required services easily. In [4] [12], the authors proposed DHT systems (eg. Chord and CAN) as the overlay network, which may lose the semantics of keys and could not support range query and prefix search well. Enhanced solutions can support semantic search through employing semantic service description language (eg. OWL-S and WSMO) which would help to eliminate this shortcoming. However, these solutions didn’t keep semantic context at overlay layer but at application layer; In addition, they ignore structure information of services which may hold the underlying semantics of services. In [13], the authors adopted a locality-preserving mapping algorithm called Space Filling Curves (SFC) to support multi-keywords search at overlay layer. It’s also not suitable for xml-based documents’ search. In our paper, we scheme Skip Graph based service overlay to try to support range query, prefix search and semantic matching at overlay layer.

3. WSDL Partition

Web services description language WSDL is xml-based document, whereas current P2P systems including Skip Graph can only support key publishing and distribution. We should split the service document into keys while keeping its tree structure and context before service publishing.

We define XML fragments [3] split from service description as keys which is expressive enough to cover most service discovery needs. Our partition algorithm is based on the tree structure of WSDL document by pre-order traversal. Each fragment is expressed as context = "term" where term is the value of the attribute in WSDL document and context is the tree path of this attribute. Thus the assignment of (key, value) pairs in Skip Graph is based on the following rules:

1) The root tree node of WSDL document has key "/" and value "null";
2) All other fragments use the tree path in WSDL /element1/.../elementn/@attr = "attrValue" as the key, and the service Access Point or semantic descriptions as values.

Considering that the number of elements and attributes vary with different WSDL documents. We give the following principles of fragment partition:

1) Reserves attribute "message" of element "input" and "output".
2) Reserves attribute "name" of all elements.
3) Only Acquires fragments from abstract part of WSDL which represents the functionality of the service.

According to these principles, we would get at least six context for each WSDL document. Table 1 gives these context acquired from the WSDL document.

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Def</td>
<td>/definitions/@name</td>
</tr>
<tr>
<td>Ctype</td>
<td>/definitions/types/complexType/@name</td>
</tr>
<tr>
<td>If</td>
<td>/definitions/interface/@name</td>
</tr>
<tr>
<td>Op</td>
<td>/definitions/interface/operation/@name</td>
</tr>
<tr>
<td>In</td>
<td>/definitions/interface/operation/input/@message</td>
</tr>
<tr>
<td>Out</td>
<td>/definitions/interface/operation/output/@message</td>
</tr>
</tbody>
</table>

The syntactic structure of WSDL is changeless, so to facilitate the keys’ storage and query, we give each context.
an abbreviation as shown at Table 1. All keys will be created from these context.

For example, we would get seven fragments from the service OrderService shown at Table 2:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Def=&quot;PurchaseOrder&quot;</td>
</tr>
<tr>
<td>2</td>
<td>Ctype=&quot;processPurchaseOrderRequest&quot;</td>
</tr>
<tr>
<td>3</td>
<td>Ctype=&quot;processPurchaseOrderResponse&quot;</td>
</tr>
<tr>
<td>4</td>
<td>If=&quot;PurchaseOrder&quot;</td>
</tr>
<tr>
<td>5</td>
<td>Op=&quot;processPurchaseOrder&quot;</td>
</tr>
<tr>
<td>6</td>
<td>In=&quot;processPurchaseOrderRequest&quot;</td>
</tr>
<tr>
<td>7</td>
<td>Out=&quot;processPurchaseOrderResponse&quot;</td>
</tr>
</tbody>
</table>

After we apply the above mechanisms to acquire service fragments, we could easily distribute these fragments as (key, value) pairs into Skip Graph overlay network while keeping the structure of service descriptions.

4. Overlay Model

After acquiring service fragments, we put forward two models of Skip Graph overlay network for WSDL publishing: planar model and tree model. We introduce these two models in detail at section 4.1 and 4.2.

4.1 Planar Model

Planar model distributes all WSDL fragments into one Skip Graph. This model almost has the same algorithm with original Skip Graph except that we support fragments with the same keys from different WSDL documents by keeping the different values of AccessPoint in a list in alphabetic order.

Figure 1 gives an example of route messages from the peer with key Out = "processPurchaseOrderResponse" to find the peer with key If = "PurchaseOrder".

Planar model is simple but not efficient. If there are N WSDL documents and total n keys split from these documents, it will take Log(n) time and Log(n) messages for WSDL query, insertion and deletion.

4.2 Tree Model

The difference between planar mode and tree model is that tree model distributes fragments of one WSDL document into different Skip Graph according to the context of the keys. Thus there would have two circumstances to query on tree model: inner Skip Graph inquiry and outside Skip Graph inquiry. Inner Skip Graph inquiry is the same as the inquiry in planar model. As for outside Skip Graph inquiry, we give the following definitions and algorithm to facilitate service discovery.

**Definition 1:** Data Skip Graph (DSG) is to represent those have keys with the context like /e₁/e₂/.../eₙ/@attribute; Virtual Skip Graph (VSG) is to represent those have the context like /e₁/e₂/.../eₙ.

Each Data Skip Graph would store the keys with the same context. Whereas each Virtual Skip Graph would not store any keys, they just survive to route the messages and organize the tree model.

**Definition 2:** If there exists two path P₁ = /e₁/e₂/.../eₖ and P₂ = /e₁/e₂/.../eₖ/eₖ₊₁. We call the VSGₖ created from P₁ is the parent agent of the VSGₖ₊₁ created from P₂. If P₃ = /e₁/e₂/.../eₖ/@a₁, we call the DSGₖ created from P₁ is the parent agent of the DSGₖ created from P₃.

Obviously, the root VSG created from "/" is the parent agent for all Skip Graph. Figure 2 gives the tree model organized by the above definitions.

Each Data Skip Graph would build up an agent peer (AP)
with key $/e_1/e_2/\ldots/e_n/@attribute=aValue''$ to transfer messages to another Skip Graph from its inner peer node. Each VSG stores two routing information: other VSG peer node ID and its children DSGs’ AP node ID. The following gives the inquiry operation in detail:

Assumes the query $Q : /e_1/e_2/\ldots/e_n/a_i=aValue''$ is initialized at peer node with key $/e_1/e_2/\ldots/e_1/a_j=queryStartPeer''$. 

1) If $n=l$ and $i=j$, then the destination key and start key are in same Data Skip Graph, call inquiry operation in Skip Graph to get the needed key.

2) If $n=l$ and $i\neq j$, the query is sent to the agent peer $/e_1/\ldots/e_n/@a_i=aValue''$, then to the VSG $/e_1/\ldots/e_n$. This VSG is also the parent agent of $/e_1/\ldots/e_n/@a_j$. We can get the agent peer node ID, and route to $/e_1/\ldots/e_n/@a_j=aValue''$ at this peer, we all inquiry operation in Skip Graph to get the needed key.

3) If $n\neq l$, the query message first is sent to the agent peer $/e_1/\ldots/e_n/@a_i=aValue''$, then to the parent Skip Graph $/e_1/\ldots/e_n$. At this Skip Graph, we can get the node ID of the VSG $/e_1/\ldots/e_1$, route the message to this Skip Graph, and to $/e_1/\ldots/e_1/@a_j=aValue''$ correspondingly. Finally we call inquiry operation in Skip Graph to get the required key.

Figure 3 gives an example of outside inquiry:

5. Prefix Search

As we described above, there would be $N$ WSDL documents and $n$ keys split from these documents where $n \gg N$. If we take each one as the key of the peer node in Skip Graph, there would create overmuch peer nodes which could not support range query and prefix search well. In this paper, we cluster those that have the same longest prefix into the same bucket and take the smallest key as the representative key of this bucket. The bucket mechanism would cluster similar keys with same prefix and thus supports prefix search and range query on keys. In the following we expound the policy of bucket creation.

**Definition 1:** The universal numerical value of the key is expressed as

$$ Uni(key) = \sum_{i=0}^{\|key\|−1} d^{Max_s−i−1} * Num(key[i]) $$ (1)

Where $\| key \|$ is the length of the key, $Num(key[i])$ is the numerical value of the $i$th character of the key. $Max_s$ is the maximum length of key, $1 \leq\| key \| \leq Max_s$. So a key can be converted to a universal $d$-radix integer and the similarity between two keys can be expressed as $Sim(key_1, key_2)$, which is the absolute difference between numerical value of the keys.

**Definition 2:** The normalized similarity between two keys:

$$ Sim(key_1, key_2) = \frac{|Uni(key_1) - Uni(key_2)|}{Uni(Z) - Uni(0)} $$ (2)

Where $Z$ means the string "zz...zz" with the length $Max_s$, 0 means '0'. Obviously we get $0 \leq Sim(key_1, key_2) \leq 1$. The more similar two keys are, the more smaller the value of $Sim(key_1, key_2)$ is.

We set a threshold $\sigma$ to control the creation of new bucket and make $RK_i = 2\sigma(i \geq 1)$. When the similarity between the new inserted key and the representative key(RK) of a bucket is smaller than $\sigma$, the new key would be inserted into this bucket; Else if this new key cannot be inserted
into any bucket, a new bucket should be created, representative key of the bucket is calculated making the similarity $Sim(newkey, RK_{new}) \leq \sigma$. The following gives the process of new key insertion.

1) Initialize the system, set the initial value of $Max_s$ and $\sigma$.

2) When a new key is added into Skip Graph, calculate $Sim(key_{new}, 0)$, find two buckets whose associated $RK$ are more similar to the new one. In any case, there exists $i(i > 0)$ that satisfies $2i\sigma \leq Sim(key_{new}, 0) \leq 2(i + 1)\sigma$.

3) If $Sim(key_{new}, 0) - 2i\sigma \leq 2(i + 1)\sigma - Sim(key_{new}, 0)$, then check whether bucket with $RK = 2i\sigma$ exists. If exists, we just insert this new value to this existing bucket, else create a new bucket with $RK = 2i\sigma$ and insert the new key into this bucket.

4) If $Sim(key_{new}, 0) - 2i\sigma > 2(i + 1)\sigma - Sim(key_{new}, 0)$, then check whether bucket with $RK = 2(i + 1)\sigma$ exists. If exists, we just insert this new value to this existing bucket, else create a new bucket with $RK = 2(i + 1)\sigma$ and insert the new key into this bucket.

For example, we assume $Max_s = 15$ and $\sigma = 7.716e^{-4}$. If we insert the key $Def = "PurchaseOrder"$ into the Skip Graph, we have $Sim(Key_{purchaseOrder}, 0) \approx 0.7182$. We get $i = 465$ and insert this key to the bucket with $Sim(RK, 0) \approx 0.7176$.

The location of the new inserted key is determinable. That is, we can calculate RK of the bucket which the key belongs to in advance. This determinable mechanism certainly will promote the efficiency of key insertion and inquiry due to early computation of RK and $Sim(key, 0)$.

When a query is originated, we can also calculate the similarity between the destination key and ‘0’, and thus get the bucket that the destination key would belong to.

6. Experimental Evaluation

To evaluate the models proposed above, we built a prototype based on Skip Graph overlay, which has access to 3526 WSDL documents from web or the main authoritative UDDI repositories. Using the WSDL partition algorithm introduced in Section 3, we extracted about 494632 non-repetitive fragments which would be distributed to different Skip Graph overlays including planar model, tree model and tree model with prefix search as resource keys.

According to the partition algorithm described in Section 3, only the abstract part of WSDL will be extracted as key, which is expected to be able to represent the crude WSDL efficiently. Figure 4 gives the evaluation of the partition method. We computed the similarity of crude fragments (including concrete, abstract) and fragments acquired by our partition algorithm using TF-IDF function. As shown in Figure 4(a), number of fragments gained by our algorithm account for little proportion in the whole WSDL documents. However, in Figure 4(b),(c),(d), we can still see that the similarity between them is high, that is about 0.7638 for the average similarity, which indicates that the partition algorithm maintained the core information of WSDL document effectively.
Our experiments showed that the combination of tree model and prefix-based clustering mechanism provides comparable performance on service publishing and discovery.

In future work, we plan to expand tree model to support other service description languages (such as OWL-S, WSMO and WSDL-S) dynamically. In addition, we would like to realize our prototype system to meet the actual requirements of service publishing and discovery at Internet.

References