SRC: A Service Registry on Cloud Providing Behavior-aware and QoS-aware Service Discovery

Hao-peng CHEN[1,2], Shao-chong LI[1]

[1] School of Software
Shanghai Jiao Tong University
Shanghai 200240, P.R. China

[2] School of Computer Science
Georgia Institute of Technology
Atlanta 30332, USA

E-mail: chen-hp@sjtu.edu.cn, lee.shaochong@gmail.com

Abstract—**Aiming to discover the most suitable service cater to the discovery request of service consumer which includes functional requirements and nonfunctional requirements, this paper proposes a service registry model named as SRC (Service Registry on Cloud) which is an extension of the keywords based service registry model and deployed as a cloud application to provide behavior-aware and QoS-aware service discovery services. SRC stores the semantic descriptors of Web Services and the feedbacks of dynamic status of QoS of Web Services as GFS files in a cloud, and uses MapReduce mechanism to process these files. The running results of an instance of SRC deployed in an experimental environment have shown that SRC is effective and feasible.**

Keywords-service registry; behavior-aware; QoS-aware; cloud; service discovery

I. INTRODUCTION

With cloud computing, service providers enjoy greatly simplified software installation and maintenance and centralized control over versioning and offload these problems to cloud computing provider [1]. The diversity of the services hosting on the infrastructures or platforms provided by clouds brings a challenge: how to discover the most suitable service cater to the discovery request of service consumer which includes functional requirements and nonfunctional requirements.

For the functional requirements, the approach of improving service discovery involves adding semantics to the Web service descriptions and then registering these descriptions in the registries [2]. SWS (Semantic Web Services) emerged as a distinct research field, and a large number of initiatives began not long thereafter, including OWL-S [3] and WSDL-S [4]. As a result, it requires that not only WSDLs but also semantic description of services to be registered into service registry for behavior-aware service discovery.

For the nonfunctional requirements, the information on quality of service is the basis for processing the service discovery requests. There are two kinds of QoS information: static rating of QoS and dynamic status of QoS. WSLA [5] is an example for describing static quality rating. To get the dynamic status of QoS, we need to monitor and measure the services at run-time.

To support both behavior-aware and QoS-aware service discovery, the service registry should have enough storage space to store the necessary data, be a data-intensive application for processing service discovery requests, and have high scalability to cater for the growing business requirements. With the emergency of cloud computing, the barrier of being a service registry provider is pulled down [6]. This paper proposes a service registry model named as SRC which is an extension of the keywords based service registry model and deployed as a cloud application to provide behavior-aware and QoS-aware service discovery services.

The remainder of the paper is structured as follows. Section II briefly summaries the related works; Section III gives the framework of the SRC; Section IV describes how SRC processes the requests of service discovery; Section V describes an instance of SRC; and conclusion in Section VI.

II. RELATED WORK

Aware of the conspicuous deficiency of key-words based search, researchers are attempting to design semantics matching search by add semantic description into the service descriptors. In [7], Paolucci et al. proposed an ontology-based solution, which matching Inputs/Outputs of Services by evaluating their semantic similarity between them according to the hierarchical concept relationships defined in an ontology tree. In [2], K. Verma et al. use an ontology-based approach to organize registries into domains, enabling domain based classification of all web services.

Researchers adopt two ways to support QoS-aware service discovery: to add static QoS rating and to obtain run-time status of QoS. Kyriakos Kritikos et al. developed an OWL-S based solution, called OWL-Q, to describe static QoS rating [8]. MDS for GT4 aims to monitor and discover resource in a grid environment [9]. IBM' Tivoli software suite also provides the capability to monitor the status of web service hosted in Websphere [10]. In [11], we design a P2P based service registry, named as QMC, to store such information and provide the QoS-aware service discovery by analyzing the stored information. The database structure and the algorithm of QoS-aware service discovery of QMC are used in SRC.

III. FRAMEWORK OF SRC

As shown in Figure 1, SRC is a cloud application. SRC takes the responsibility of collecting real-time status of dynamic quality of services and supports behavior-aware and QoS-aware service discovery, which is an extended UDDI-complied service registry. SRC has three ports, namely Service Publish Port, Service Discovery Port, and Service Feedback Port. The Service Publish Port is an extended UDDI-complied port, which means the service providers can not only register WSDL files but also semantic description into the service registry. With the Service Discovery Port, also an extended UDDI-complied port, service consumers can discover services not only by keyword-based constraints but also by semantic-based constraints and QoS-based constraints. The Service Feedback Port is a new port which is designed for collecting feedbacks from Service Customers and Service Providers. As the portal of service registry, all the three ports are deployed on the Cloud Controller, which is the single access point of a cloud.

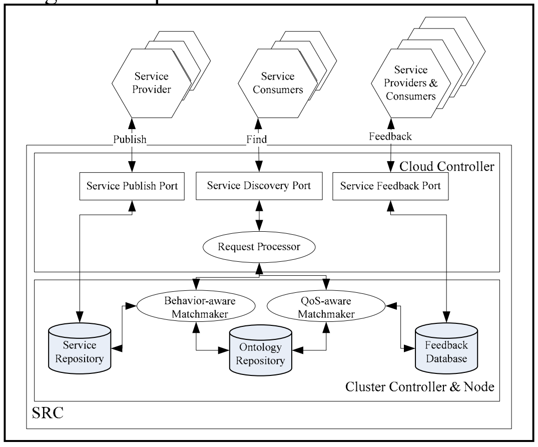

Figure 1. Architecture of the Service Registry

The Service Repository holds the static description of services, and the Feedback Database stores the dynamic feedbacks. Since both of their sizes keep growing, we use GFS[12] as the file system to store and manage them, which means they will be divided into chunks and stored on multiple nodes of a cloud. Unlike the former tow repositories, the Ontology Repository will be seldom changed once it is established. It stores all the ontologies, including Domain Ontology which is used to calculate the similarity of the inputs/outputs of two services, Predicate Ontology which is used to calculate the similarity of the behaviors of two services, and QoS Ontology which is used to describe the features of each QoS attribute. Both of the Behavior-aware Matchmaker and QoS-aware Matchmaker will access the Ontology Repository to accomplish their tasks. Allowing for the access efficiency, we duplicate the Ontology Repository on each node which has a chunk(s) of either Service Repository or Feedback Database. Thus, both the Matchmakers can access the Ontology Repository locally rather than remotely.

When a Service Consumer proposes a service discovery request, the request is dispatched to the Request Processor which forks a certain number of instances of Behavior-aware and QoS-aware matchmakers, divides the request into functional requirements and QoS requirements, and respectively passes them to the instances of Behavior-aware and QoS-aware matchmakers. The Behavior-aware and QoS-aware matchmakers use MapReduce[13] mechanism to respectively find a set of function-matched services and a set of QoS-matched services. The Request Processor calculates the intersection of the two result sets returned by Behavior-aware and QoS-aware matchmakers. The resulting intersection is the final result which is sent back to users. Since the Request Processor is responsible for forking instances of Matchmakers and merging the result sets, it also should be located on Cloud Controller. The instances of Behavior-aware and QoS-aware matchmakers dynamically forked by Request Processor are running on the Cluster Nodes or Cluster Controllers.

IV. Service Matchmaking

A. Behavior-Aware Semantic Matchmaking

When a service is registered into SRC, its semantic descriptor will be submitted to SRC with WSDL descriptor together. We store them in a cloud and use GFS to manage them. We must guarantee that the records of inputs, outputs, predicates, and constraints of a service shall be stored in the same node of the cloud and the OWL_Class table shall be stored in every node of the cloud. Thus, even if the tables are divided into chunks and each chunk will be stored in an exclusive node, when we want to access all the semantic information of a service, we needn't do any inter-node query. Since OWL_Class table is stored in every node, we need synchronize its multiple copies on all nodes. The synchronization can be done by any DBMS.

The algorithm for matching any parts included in a service functional description model shall compute semantic similarity of two targets (concepts, predicates, or constraints). For the sake of behavior-aware, we focus on concept similarity and predicate similarity. Similar with nouns grouped by Class sets in ontology, we organize verbs, more specifically predicates, into Property sets in domain specified ontology. Then, we interlink property sets with their relationships as *subPropertyOf* and build a taxonomy tree describing the subsumption relationships between all the properties of the ontology [14]. In SRC, this algorithm is used in behavior-aware matchmaker as the workers of MapReduce mechanism.

The functions of behavior-aware matchmaking would be written in the code similar to the following pseudo-code:

```
function map (String key, Collection value):
  // key: functional requirements
  // value: the registered services
  for each service s in value:
    $\mu$= $Sim_{Behavior}$(s, key) // compute the similarity
                                  //between s and key
    if $\mu$>= $\alpha$ // $\alpha$ indicates the acceptable similarity
          // specified by user.
       // save the pair < s, $\mu$> on a position of local
       //disk according to the partitioning function.
       EmitIntermediate(s, $\mu$);
    endif
  endfor
end function
```

```
    function reduce( Float key, Iterator value):
        // key: a similarity between α and 1
        // value: a list of pairs < s, μ>
        Collection result = {};
        for each < s, μ> in candidates:
          result = result ∪ {s};
        endfor
        Emit(result);
    end function
```

The map function emits each acceptable service plus an associated similarity. The partitioning function divides the range [α, 1] into R_b equal portions (R_b is specified by users) and each portion is mapped onto a local disk position. The pair <s, μ> generated by map function will be saved on the position of local disk onto which the portion covering μ is mapped. The reduce function merges together all the services with the similarity in same portion.

B. *QoS-Aware Service Matchmaking*

To avoid subject feedbacks and be independent of web service hosting environment, we need a monitor to send the feedbacks. Figure 2 shows the structure of such a monitor, called QoS Spy [15]:

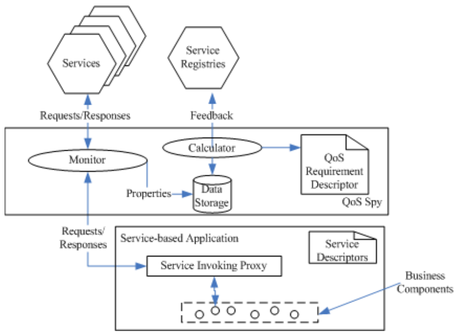

Figure 2. The Structure of QoS Monitor

In the structure shown in Figure 2, the QoS Spy is composed of a Monitor, a Calculator and a lightweight Data Storage. It runs in the hosting environment of service-based application. The Monitor acts as an interceptor of each service invocation and records information like exact intercepting time and target service. The Calculator calculates the average processing time, average request frequency, and average other stores them in the Data Storage.

As a part of QoS Spy, the QoS interceptor usually runs on the hosting environment of service-based application. As Service Providers and Service Consumers both need the QoS interceptor, it also can separately run as an independent tool in the hosting environment of services to feedback the global real-time status of dynamic qualities of services.

The functions of QoS-aware matchmaking would be written in the code similar to the following pseudo-code:

```
    function map (String key, Collection value,):
      // key: nonfunctional requirements on QoS attributes
      // value: the registered services
      for each service s in value:
        Bool match = true; //match indicates if s matches key.
        Vector stat = NULL; // stat holds the attributes of s.
        for each attribute q in key:
          λ= smooth(q)  //λ is the smoothed value of specified
                        //number of feedbacks of q
          if λ meet the requirement on q
            put(stat, λ); // put λ into stat.
          else
            match = false;
            break; //s doesn't match q
          endif
        endfor
        if match = true
          // save the pair < s, stat> on a position of local disk
          // according to the partitioning function.
          EmitIntermediate(s, stat);
        endif
      endfor
    end function

    function reduce( String key, Iterator value):
      // key: a quality attribute
      // value: a list of pairs < s, stat>
      Collection result = {};
      for each < s, stat> in candidates:
        result = result ∪ {s};
      endfor
      Emit(result);
    end function
```

The map function emits each acceptable service plus a set of smoothed value of all quality attributes. The partitioning function divides each local disk into R_q portions (R_q is specified by users) each of which represents a range of distance between vector *stat* and query vector containing all the nonfunctional requirements of users. The reduce function merges together all the services with the distance in same portion.

C. *Process of Request Processor*

When SRC receives the service discovery request on the Service Discovery Port, it dispatches the requests to the Request Processor. The latter decomposes the request into functional requirements part and nonfunctional requirements part, and dispatched them to Behavior-aware matchmaker and QoS-aware matchmaker separately. The steps of process of Request Processor are as followings [13]:

1. Request Processor respectively splits the Semantic Descriptor Database and Feedback Database into M_b and M_q pieces of specified MB per piece.
2. Request Processor starts up instances of Behavior-aware and QoS-aware Matchmakers on cluster nodes of cloud. The special instances, master Behavior-aware matchmaker B_m and master QoS-aware matchmaker Q_m, are running inside Request Processor.
3. The rest are workers that are assigned work by the B_m and Q_m. There are M_b and M_q map tasks and R_b and R_q reduce tasks to assign. The B_m and Q_m pick idle workers and assigns each one a map task or a reduce task. The workers are running in VMs on

cluster nodes of cloud. Each cluster node can run one or more VMs to run workers.

4. A worker who is assigned a map task described in Section IV.A and IV.B. The intermediate key/value pairs produced by the Map function are buffered in memory.
5. Periodically, the buffered pairs are written to local disk, partitioned into R (R_b and R_q) regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.
6. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers and send them to Reduce function. The output of the Reduce function is appended to a final output file for this reduce partition.
7. When all map tasks and reduce tasks have been completed, the master wakes up the Request Processor. At this point, the Request Processor calculates the intersection of the result sets returned by B_m and Q_m and returns it as the response to the service discovery request.

V. An Instantance of SRC

We establish a 10-computer environment for instantiating SRC in which we runs one CLC, three CCs, and six NCs. We simulate the semantic descriptors and feedbacks of more than 10,000 web services to accumulate necessary data. The running results of this instance of SRC have shown that SRC can provide effective supports for behavior-aware and QoS-aware service discovery.

The efficiency of this instance of SRC has not been considered for two reasons. The first one is that the hosting environment we establish is an experimental one which is quite different from the true cloud platform. The results from this environment do not reflect the real efficiency of SRC. The second one is that the data we simulate are not massive enough to check the efficiency of SRC. MapReduce is much more efficient than other query methods when the data to be processed is massive. It is difficult for us to accumulate such massive data in an experimental environment.

VI. Conclusion

Aiming to discover the most suitable service cater to the discovery request of service consumer which includes functional requirements and nonfunctional requirements, this paper proposes a service registry model named as SRC which is an extension of the keywords based service registry model and deployed as a cloud application to provide behavior-aware and QoS-aware service discovery services. SRC stores the semantic descriptors of Web Services and the feedbacks of dynamic status of QoS of Web Services as GFS files in a cloud, and uses MapReduce mechanism to process these files.

The running results of an instance of SRC deployed in an experimental environment have shown that SRC is effective and feasible. In next work, we will deploy it into a commercial cloud and analyze its efficiency.

Acknowledgment

As a visiting scholar of Georgia Institute of Technology, Hao-peng Chen thanks Georgia Institute of Technology and Prof. Ling Liu, a Professor of College of Computing at Georgia Institute of Technology, for their supports.

This paper is also supported by the Project of Daystar of Shanghai Jiao Tong University.

References

[1] Michael Armbrust, Armando Fox, et.al, *Above the Clouds: A Berkeley View of Cloud Computing*, http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf, 2009

[2] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar and J. Miller, *METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services*, Journal of Information Technology and Management, Special Issue on Universal Global Integration, Vol. 6, No. 1 (2005) pp. 17-39. Kluwer Academic Publishers.

[3] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, et al. *OWL-S: Semantic Markup for Web Services*, http://www.w3.org/Submission/OWL-S/, 22 Nov. 2004.

[4] Rama Akkiraju, Joel Farrell, John Miller, Meenakshi Nagarajan, Marc-Thomas Schmidt, et al. *Web Service Semantics - WSDL-S Version 1.0*, http://www.w3.org/Submission/WSDL-S/, 7 Nov. 2005.

[5] Heiko Ludwig, Alexander Keller, Asit Dan, Richard P. King, Richard Franck, *Web Service Level Agreement (WSLA) Language Specification*, http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf, 2003/01/28

[6] VOGELS, W. *A Head in the Clouds—The Power of Infrastructure as a Service*. In First workshop on Cloud Computing and in Applications (CCA '08) (October 2008).

[7] Massimo Paolucci, Takahiro Kawamura, Terry R.Payne, and Katia Sycara, *Semantic Matching of Web Services Capabilities,* The First International Semantic Web Conference on The Semantic Web(ISWC 2002), 2002, pp.333-347.

[8] Kyriakos Kritikos, Dimitris Plexousakis, *OWL-Q for Semantic QoS-based Web Service Description and Discovery*, First International Joint Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web, PP123-137, 2007

[9] *GT Information Services: Monitoring & Discovery System (MDS)*, http://www.globus.org/toolkit/mds/

[10] *Tivoli Monitoring,* http://www-01.ibm.com/software/tivoli/products/monitor/

[11] Siming Xiong, Haopeng Chen, *QMC: A Service Registry Extension Providing QoS Support*, Proceedings of 2009 International Conference on New Trends in Information and Service Science (NISS 2009), PP.145-151, Beijing, China, 2009.6.30-2009.7.2

[12] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, *The Google File System*, ACM SIGOPS Operating Systems Review Volume 37 , Issue 5 (December 2003) Pages: 29-43

[13] Jeffrey Dean and Sanjay Ghemawat, *MapReduce: Simplied Data Processing on Large Clusters*, Proceedings of 6th Symposium on Operating Systems Design & Implementation (OSDI '04) , pp. 137-150. Dec. 2004.

[14] Juncheng Yang, Haopeng Chen. *A Behavior-Aware Matchmaking Model for Semantic Web Services Discovery*. Proceedings of 6th International Conference on Networked Computing and Advanced Information Management. PP.183-188, Seoul, Korea, 2010.8.

[15] HAO-PENG CHEN, CAN ZHANG, GUANG YANG, A *Mechanism for Managing and Discovering Services Based on Dynamic Quality of Services*, Journal of Networks (JNW), PP.888-895, VOL. 5, NO. 8, AUGUST 2010.