QMC: A Service Registry Extension Providing QoS Support[1]

Siming Xiong, Haopeng Chen
School of Software, Shanghai Jiao Tong University, P.R.China, 200240
simingxiong@gmail.com, chen-hp@sjtu.edu.cn

Abstract

With the rapidly growing number of Web services throughout the Internet, the limitations of centralized architecture and the neglect of QoS support have severely restricted service registries' ability to publish and discover Web services. We propose a P2P service registry extension named QMC to solve these problems. QMC provides comprehensive support on QoS such as storing QoS feedbacks, managing QoS data, handling QoS requests. Moreover QMC is a system with high scalability and load-balance.

1. Introduction

Service Registry plays an important role in Web services (WS) discovery. Without it, service providers have to pay more prices to advertise their services while service consumer cannot discovery a required service in an efficient manner.

In recent years, several UDDI Business Registries (UBR) established by IBM, SAP, Microsoft have emerged. However, none of them are pervasively used according to the survey in [1].The main reason is that they only considered functional criteria. Unfortunately, there are now thousands of functional- equivalent Web services disseminated throughout the Internet and it will be a time-consuming task for service consumers to find a satisfying WS. As a result, a certain subset of all possible non-functional properties that may affect the quality of the service collectively referred to as QoS has been taken into account. By doing so, QoS has substantial impacts on users' expectations of services and can be efficient as a discriminating factor among web services providing equivalent functionalities.

Moreover, UBRs are based on centralized architecture. Although centralized registries can provide effective methods for the discovery of Web services, they suffer from problems associated with having centralized systems such as a single point of failure, and bottlenecks. In response to this problem, the majority of researchers suggest a P2P solution. A P2P overlay network provides an infrastructure for routing and data location in a decentralized, self-organized environment in which each peer acts not only as a node providing routing and data location service, but also as a server providing service access. It not only avoids the drawbacks of centralized registry, but also allow service registry to share WS information with high scalability. Recently proposed P2P protocols include CAN[2], Pastry[3], Chord[4]. All of them support looking up data by a unique key.

Based on the above, we propose a system called QMC which extends the existing service registries to support queries based on QoS requirements. Besides, QMC takes the advantages of P2P systems. The rest of this paper is organized as follows: Section 2 surveys the related work. Section 3 makes some assumptions and illustrates our changes to the SOA model. Section 4 presents the internal architecture of our system and Section 5 explains the process of QoS data distribution and QoS query by exemplification. Section 6 draws the conclusions and describes our future work.

2. Related Work

Although QoS plays an important role in Web service discovery, a widely-accepted standard specifying Qos metrics and measurement has not existed yet. Many researchers have proposed their own Qos definitions [5][6][7]. These Differences in QoS concepts and

[1]This paper is supported by the National High-Tech Research Development Program of China (863 program) under Grant No. 2007AA01Z139.

978-0-7695-3687-3/09 $25.00 © 2009 IEEE
DOI 10.1109/NISS.2009.54

IEEE computer society

measurement may result in confusion and misleading in the process of Web service discovery. Therefore, another more acceptable way is to define a rich QoS description model. Many different ontologies have been proposed such as OWL-Q [8], DAML-QoS [9], QoSOnt [10], WSMO-QoS [11]. These ontologies explore a way to define different QoS definitions in a QoS description model.

Regarding the P2P service registry, the majority of researchers focus on functional criteria. In [12], the author proposes a web service index system based on Chord. In this system, all data elements are described using a sequence of keywords. These keywords form a multidimensional keyword space. The author uses a Hilbert-Space Filling Curve to map the n-dimensional keyword space to a one-dimensional indexing space and hash it onto the underlying node in the Chord circle. Similarly, [13] proposes A scalable Web Service discovery Architecture based on P2P. This system converts the service description files (WSDL files) into a node-value tree and generates a hash value from the tree nodes. After that, the service description will be inserted into the Peer-to-Peer overlay network. Nevertheless, another approach attempts to provide QoS support for the service registry. [14] proposes a P2P semantic service discovery architecture with QoS support. Unfortunately, it only provides a QoS ranking algorithm without any reference to QoS queries (e.g. find all the services with response time less than 500 ms).

In summary, all the above P2P systems either do not consider QoS or provide insufficient QoS support. So in this paper, we will discuss how our QMC system provides an enhanced QoS query capability.

3. Assumptions and SOA Model

In order to focus on the process of discovering WS through QoS query, we make the following assumptions:

1. There has already existed a service registry for QMC to extend. So we do not take functional criteria into consideration and focus on QoS only.

2. Every service consumers and service providers have a unique identifier (UID).

3. The same WS in the service registry and in QMC have the same service identifier (SID).

4. All the QoS metric should be measurable in case of a subjective evaluation of QoS.

5. Both service consumers and service providers monitor the status of the services and send credible QoS feedback to QMC. So we do not consider malicious reporting and collusive cheating of service consumers and service providers. This assumption can easily be removed. Some colleagues of mine are working on it and have made some progress.

Obviously, all the above assumptions are reasonable. With these assumptions, we can focus on the procedure of QoS query.

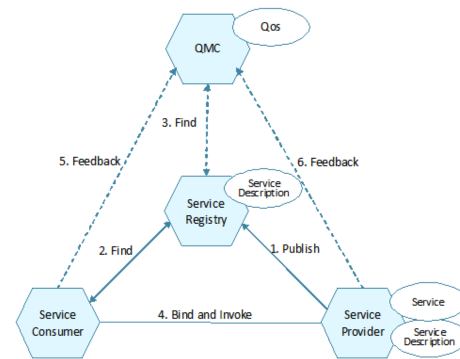

Figure 1. SOA Model

Figure1 illustrates our SOA model. We make some changes to original SOA model. All the solid lines in this figure represent the operations which have already been implemented in the original SOA model, while all the dashed lines stand for the operations proposed by our model. A typical process of our model is as follows: various service providers publish their service descriptions onto the service registry (1), and service consumers query for services with certain functional and QoS requirements (2). The service registry handles the functional part of the query and builds QoS request combining both the functional query results and QoS requirements. QMC receives the QoS request (3) and finds the required services. The results are returned to the service consumer who may invoke one of the found services (4).The service consumer monitors the status of the services and sends QoS feedbacks to QMC(5). Additionally,

service providers can also express QoS feedbacks which they could obtain from the service to QMC (6).

4. Architecture

To fulfill the functions of QMC as mentioned in Section 3, each QMC peer in the system should be responsible for storing QoS feedbacks, managing QoS data, handling QoS request.

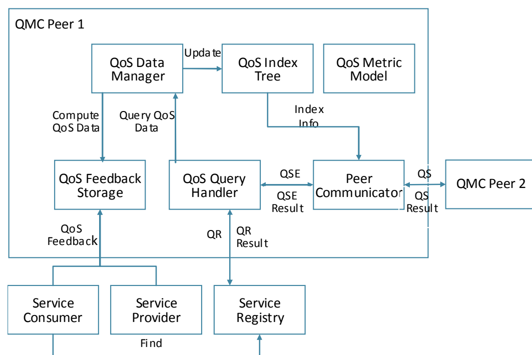

Figure 2. The internal architecture of a QMC peer

Figure 2 shows the internal architecture of each QMC peer. It is composed of 6 components: QoS Feedback Storage, QoS Metric Model, QoS Data Manager, QoS Index Tree, QoS Query Handler, Peer Communicator.

4.1 QoS Metric Model

As we mentioned in Section 2, a widely-accepted standard specifying Qos metrics and measurement has not existed yet. Therefore, it is essential for us to adopt a formal QoS ontology to model different QoS metrics.

Compared with other QoS ontologies we refered to in Section 2, OWL-Q provides a more comprehensive description of QoS metrics. The Metric Facet of OWL-Q describes all the appropriate classes and properties used for defining QoS metrics. A QoSMetric is separated into static and dynamic metrics. A StaticQoSMetric is computed only once to produce a value while a DynamicQoSMetric is computed repeatedly according to a Schedule to produce values that change over time. A DynamicQoSMetric can be further separated into a simple QoS metric measured by a MeasurementDirective or a complex one derived from other metrics with the help of an OMFunction.

Moreover, OWL-Q strongly supports us to describe various unit types and values types for QoS metrics. The Unit Facet describes the unit of a QoS metric. It also contains concepts that are used to convert values of equivalent metrics having different units. The QoSValueType Facet describes the value types a QoS metric can take, such as string, number, range, list, boolean.

Based on the above considerations, we adopt OWL-Q as our QoS metric model.

4.2 Peer Communicator

Peer Communicator is the QMC peer's interface to other peers. It is used for routing and locating QoS data. In this work, we implement Peer Communicator by extending Chord algorithm to organize the network of QMC peers because of simplicity, provable correctness, and provable performance of Chord compared with other P2P data lookup protocols.

In the Chord overlay network, each peer has a unique identifier (PID) ranging from 0 to 2^m-1. These identifiers are arranged as a circle modulo 2^m, where each peer maintains information about its successor and predecessor on the circle. And each data element gets an identifier (HID) from the identifier space (0 to 2^m-1) generated by a consistent hash function. Then it is mapped to the first peer whose PID is equal to or follows the HID in the identifier space.

For the sake of routing and locating QoS data, the Chord algorithm should be extended to manipulate QoS data in the QoS Data Manager by invoking its APIs, interact with QoS Query Handler to fulfill QoS request. We will discuss how to extend the Chord algorithm in detail in Section 5.

4.3 QoS Feedback Storage

QoS Feedback Storage is the place where we store the QoS feedbacks. We assume that the feedback in the storage should be a 3-dimensional vector FB in which UID is the identifier of the user who provides the feedback, QD is a data structure containing the real-time QoS data about the identifier of the service, the QoS metric we measured and

its value, t is the time when the feedback is created.

$$FB = \langle UID, QD, t \rangle \quad (1)$$

Each FB has a HID by hashing SID in QD and is routed to a QMC peer by Peer Communicator. After receiving the FB, the peer will store it in the QoS Feedback Storage. These feedbacks are the foundation of computing statistical QoS data.

4.4 QoS Index Tree

Each QoS metric described in the metric model has an index tree. Each index tree divides the continuous value of QoS data into different ranges so that each peer only has to manage a small part of the QoS data. We make small changes to B+ Tree to implement it. Each leaf node in this tree represents a range between the keys in the internal node. It stores a data structure containing a randomly generated HID which decides the peer to store the QoS Data in the range and an integer which is the sum of services whose QoS Data is in the range. QoS Index Tree allows range queries (e.g. find all i such that $v_1 \leq i \leq v_2$) and returns all the HID in the leaf nodes whose range overlaps the query range.

Obviously, any change made to the QoS Data will update an index tree. However, we hope that each peer equally store only a small fraction of the QoS data. Therefore we need to adjust the index tree to have as many leaf nodes as possible while each leaf node have almost the same sum of services. In order to achieve this, we set a threshold to the sum of services each leaf node has. When the sum is greater than the threshold, the leaf node will split into two leaf nodes. By doing so, we can make sure each peer have almost the same load. We will discuss how to decide the threshold in Section 5.

Furthermore, we record every change we make to the index tree and spread the change through the Chord circle so as to make sure each peer has the same index tree.

4.5 QoS Data Manager

QoS Data Manager is responsible for manipulating and querying QoS Data. It statistics QoS Data from a large amount of QoS feedbacks according to the Schedule defined in the QoS Metric, distributes QoS Data to a specific QMC peer, and handles query from QoS Query Handler.

Each QoS Data is a 4-dimensional vector QD in which SID is the identifier of the service, QM is a QoS metric described by our metric model, v is the value we derived by statistical method, t is the time when the data is updated.

$$QD = \langle SID, QM, v, t \rangle \quad (2)$$

After a new QoS Data is computed, the QoS Data Manager has to distribute the data to a peer in accordance with the Chord protocol. At first, it updates the index tree according to the data. Then it invokes the API in the Peer Communicator to finish the job.

Furthermore, QoS Data Manager needs to check the QoS Data periodically in case that the data is outdated or placed at the wrong peer as a result of the changes made to the Chord network or the index tree.

4.6 QoS Query Handler

Before we explain the QoS Query Handler, we define the following concepts for the sake of convenience:

Definition 1: QoS SelectElement (QSE) is a constraint on a single QoS metric for example, response time<1 second.

Definition 2: QoS Request (QR) is a set of unprocessed constraints on some QoS metric put forward by service consumer. It is a logical expression made up of QSEs for instance, response time<1 second and availability>90%.

Definition 3: QoS Selection (QS) is a set of constraints on some QoS metric after the QoS Request have been pre-processed. For example, {response time<1 second, availability>90%}. Furthermore, it is a set of QSEs grouped by the peer where the QSE will be redirected to. As a result, QS also contains a PID indicating the peer.

QoS Query Handler is used to handle QR. When received a QR from the service registry, the Query Handler breaks down it into a set of QSEs, and passes it to the Peer Communicator where each QSE will be processed. After the result of QSE is returned, the QoS Query Handler computes the final result according to the logic of the QR.

Be aware that if the QR from service registry contains a list of SID which limits the possible services base on

functional requirement, the process still works. The only thing need to do is to add this constraint to the QSEs.

5. QoS Data Distribution and QoS Query

QoS Data Distribution and QoS Query are the most important functions of our system. QoS Data Distribution routes QoS Data to a peer in accordance with the Chord protocol. By doing so, the process of QoS Query can be accelerated by using its corresponding algorithm to locate QoS Data.

In order to implement these two functions, we mainly need to extend the Chord algorithm. In the following part of this section, we will give the pseudocode and the corresponding examples. Before that, we give the following preconditions:

Precondition 1: The result of the consistent hash function range from 0 to 63. In other words, Both PID and HID are in the identifier space from 0 to 63.

Precondition 2: There are three peers in the Chord circle whose PID are 23, 44, 61.

Precondition 3: We only consider two QoS metrics, availability (AV) and response time (RT) in this example. We assume that the values of AV are integers range from 0 to 100, and its unit is % while the values of RT are integers from 0 to infinite, and its unit is millisecond.

Precondition 4: Figure 3 illustrates the two QoS Index Trees corresponding to AV, RT. We neglect the internal nodes because it does not affect the result. We also assume that the threshold is so big that the leaf node does not split in this example.

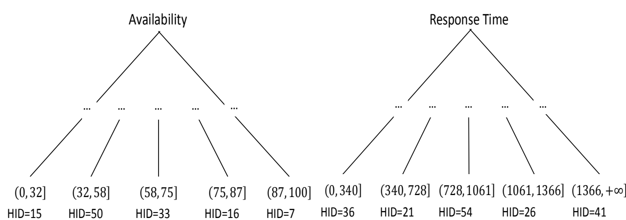

Figure 4. AV,RT index tree

Precondition 5: After computation, the QoS data is as follows:

Table 1. QoS Data

SID	QM	v	SID	QM	v
S1	AV	38	S1	RT	1214
S2	AV	45	S2	RT	324
S3	AV	64	S3	RT	779
S4	AV	92	S4	RT	422

5.1 QoS Data Distribution

Figure 4 shows the pseudocode of QoS Data Distribution. It is invoked by QoS Data Manager to route the data to a specific peer.

```
QoSDataDistribution(QD data)
1  HID hid=index(data);
2  PID pid=findSuccessor(hid);
3  PeerCommunicator p=getPeerCommunicator(pid);
4  p.insert(data);
```

Figure 4. The pseudocode of QoS Data Distribution

Take QD=<S1, AV, 38> for example.At first, we search the AV index tree according to the value 38 and acquire the HID 50. Then we invoke the findSuccessor method defined in original Chord algorithm and gain the PID 61. After that, we get the Peer Communicator p corresponding to the PID. Finally, we insert the data by invoking the insert method which actually invoking the insert method in the QoS Data Manager. The process is same to the other QoS Data, and the final result is shown in Figure 5.

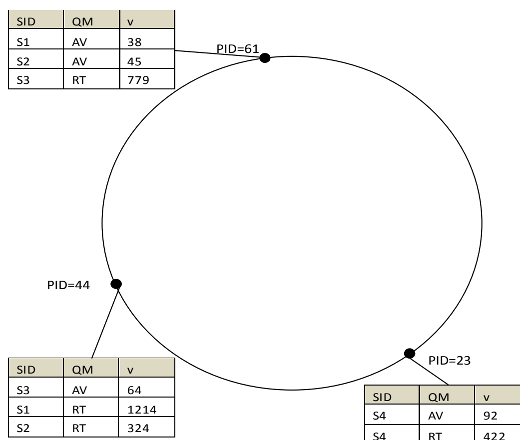

Figure 5. The final result of QoS Data Distribution

5.2 QoS Query

Figure 6 displays the pseudocode of QoSQuery which is invoked by QoS Query Handler to get the results of QSEs.

```
QoSQuery(List<QSE> qselist)
1   List<QSE> qselist=divide(request);
2   List<QS> qslist;
3   for each QSE qse in qslist
4       List<HID> hidlist = index(qse);
5       for each HID hid in hidlist
6           PID pid= find_successor(hid);
7           group(qslist,pid,qse);
8   List<QSResult> resultlist;
9   for each QS qs in qslist
10      PeerCommunicator p=getPeerCommunicator(qs.pid)
11      QSResult result = p.QoSQuery(qs);
12      resultlist.add(result);
13 return resultlist;
```

```
QoSQuery(QS selection)
1   for each QSE qse in selection
2       QSEResult result=qosDataManager.find(qse);
3   return all the result as one QSResult;
```

Figure 6. The pseudocode of QoSQuery

We will illustrate the QoS Query process base on the distribution result in Figure 5. Take the QR {AV > 80% and RT<500ms} for example. Before the QoSQuery method is invoked, the QoS Query Handler splits the request into a list of QSEs (QSE1: AV>80%; QSE2: RT<500ms).Then the QoSQuery method obtains HIDs by querying the QoS Index Tree according to the QSEs, and calculates the corresponding PIDs (QSE1: HID=7, 16→PID=23; QSE2: HID=36→PID=44, HID=21→PID=23). After that, all the QSEs will be grouped into different QSs so as to reduce the frequency of interaction with the other peer (QS1 to PID=23: {AV>80%; RT<500ms}; QS2 to PID=44: {RT<500ms}). Next, the QSs will be sent to the target peers where they will be handled by the method QoSQuery(QS selection) by the means of querying the QoS Data Manager (QS1 Result: {S4; S4}; QS2 Result: {S2}). In the end, all the result will be returned to the initial peer's QoS Query Handler and compute the final result of the QoS Request(Final Result : S4).

As we discussed before, an index tree should have as many leaf node as possible. We presume that s is the number of services registered on the QMC, q is the number of QoS metrics described in the QoS Metric Model, k_i (i=1...q) is the number of leaf nodes in an index tree. So the average number of services each leaf node has should be equal or less than the threshold.

$$\frac{q \times s}{\sum_{i=1}^{q} k_i} \leq \text{threshold} \quad (3)$$

On the other hand, if an index tree has more leaf nodes, the QoSQuery will redirect QS to more peers which may cause lower performance in query. We assume p is the number of peers in the Chord circle, $p_{redirect}$ is the number of peers where QSs will be redirected to. Obviously, $p_{redirect}$ is equal or less than the number of HIDs in the index tree which equals the number of leaf nodes. So the number of leaf nodes should be equal or less than p.

$$\sum_{i=1}^{q} k_i \leq p \quad (4)$$

Based on the above, $\text{threshold} = \frac{q \times s}{p}$ is a proper value for both load-balance and performance.

6. Conclusion and Future Work

Most of the existing service registries either do not consider QoS or provide insufficient QoS support. Therefore, we propose a P2P service registry extension named QMC to enhance their ability to manage and query QoS in this paper. Similar to other P2P system, our system scales well in terms of number of peers, search efficiency, and number of users. Furthermore, OWL-Q gives users the flexibility to model different QoS metrics.

Still, there is a lot of further work to be done. The problem how to maintain consistent between different copies of index tree in different peers has not been addressed so far in this paper. Besides, many experiments need to be performed to prove the efficiency, load-balance, robustness of our system.

Reference

[1] Eyhab Al-Masri,Qusay H. Mahmoud. Investigating web services on the world wide web. Apr 2008 Proceeding of the 17th international conference on World Wide Web

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In Proceedings of ACM SIGCOMM, pages. 161–172,San Diego, CA, 2001,

[3] A. Rowstron and P. Druschel, Pastry: Scalable, distributed object location and routing for largescale peer-to-peer systems, in Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms(Middleware), pages. 329–350.,Heidelberg, Germany, 2001.

[4] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In Proceedings of ACM SIGCOMM, pages. 149–160,San Diego, CA, 2001.

[5] Eyhab Al-Masri, Qusay H. Mahmoud. Toward Quality-Driven Web Service Discovery. IT Professional May 2008, vol.10, pp.24-28

[6] A. Mani and A. Nagarajan, Understanding Quality of Service for Web Services, http://www-106.ibm.com/developerworks/library/wsquality.html, January 2002.

[7] Menasce D.A., “QoS Issues in Web Services”, IEEE Internet Computing, November-December 2002, vol.6, pp. 72-75.

[8] K. Kritikos and D. Plexousakis. Semantic QoS Metric Matching. In Proceedings of the European Conference on Web Services (ECWS2006), IEEE Computer Society, pp.265–274, 2006.

[9] C. Zhou, L. Chia, and B. Lee. DAML-QoS Ontology for Web Services. In Proceedings of the International Conference on Web Services (ICWS04), 2004.

[10] G. Dobson, R. Lock, and I. Sommerville. QoSOnt: a QoS Ontology for Service-Centric Systems. In Proceedings of the 2005 Euromicro SEAA, 2005.

[11] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma. A QoS-Aware Selection Model for Semantic Web Services. In Proceedings of the 4th International Conference Service Oriented Computing – ICSOC 2006, LNCS, Springer Verlag, Volume 4294, pp. 390–401, 2006.

[12] CRISTINA SCHMIDT,MANISH PARASHARA.A Peer-to-Peer Approach to Web Service Discovery .World Wide Web: Internet and Web Information Systems, 7, 211–229, 2004.

[13] Yin Li , Futai Zou, Zengde Wu, and Fanyuan Ma.PWSD: A Scalable Web Service Discovery Architecture Based on Peer-to-Peer Overlay Network. APWeb 2004, LNCS 3007, pp. 291–300, 2004.

[14] Haihua Li; Xiaoyong Du; Xuan Tian: Towards P2P-Based Semantic Web Service Discovery with QoS Support.In Semantics, Knowledge and Grid, Third International Conference on 29-31 Oct. 2007 pp 358 - 361