Peer-to-Peer Based QoS Registry Architecture for Web Services*

Fei Li, Fangchun Yang, Kai Shuang, and Sen Su

State Key Lab. of Networking and Switching, Beijing University of Posts and Telecommunications
187#,10 Xi Tu Cheng Rd.,Beijing,100876, P.R.China
pathos.lf@gmail.com, {fcyang, shuangk, susen}@bupt.edu.cn

Abstract. Web service QoS (Quality of Service) is a key factor for users to evaluate and select services. Traditionally, run-time QoS of web services stores in centralized QoS registry, which may have performance and availability problems. In this paper, we propose a P2P (Peer-to-Peer) QoS registry architecture for web services, named Q-Peer. Q-Peer is an unstructured P2P system. Query of QoS is naturally achieved by getting QoS address from corresponding service description. Q-Peer has a replication based mechanism to ensure load-balance of the whole architecture. The architecture takes advantage of P2P systems to ensure its availability, performance and autonomy. We are currently implementing Q-Peer and planning to test it on Planet-Lab.

1 Introduction

Using web service technology to integrate business applications is one of the major trends of distributed computing. It is a widely known procedure that service[1] requesters discover services by functional description and select services by non-functional properties. Because service function is relatively stable throughout service lifetime, while service QoS can change frequently with time, load, network condition and many other factors, maintaining the two types of information has different system requirements and design considerations. Thus, the 2 steps are often accomplished on 2 entities respectively, called *service registry* and *QoS registry*. Centralized QoS registry has been proposed and researched before [1][2], but they are sharing some common shortcomings of centralized systems, like scalability, performance and single point failure. More importantly, because of business boundary, system scale and other limitations, centralized system may not be able to support global scale B2B interoperations. As far as we know, only Gibelin. and Makpangou [3] have considered

* This work is supported by the National Basic Researchand Development Program (973 program) of China under Grant No.2003CB314806; the Program for New Century Excellent Talents in University (No:NCET-05-0114); the Program for Changjiang Scholars and Innovative Research Team in University (PCSIRT); the Hi-Tech Research and Development Program (863 Program) of China under Grant No.2006AA01Z164.

[1] In this paper, we use *web service* and *service* interchangeably.

J. Indulska and K. Raymond (Eds.): DAIS 2007, LNCS 4531, pp. 133 – 138, 2007.

distributed QoS registry architecture but no detailed design is presented and the hash-table based QoS indexing approach is inefficient.

In past several years, peer-to-peer paradigm has gained considerable momentum as a new model of distributed computing. P2P system is created for file sharing at first, like Napster, Gnutella[4], Kazaa[5] and so on. For their scalability, autonomy and robustness, they are introduced into distributed storage and information retrieving[6]. Some applications of P2P have already contributed to web service research, as distributed service discovery[7].

In this paper, we propose our ongoing work--P2P QoS registry architecture, named Q-Peer. Q-Peer is a service QoS information storage architecture. It provides large scale QoS collecting, retrieving and monitoring services. It can work with centralized or decentralized service registry like UDDI or other P2P service discovery system. Q-Peer solving the QoS query problem by adding QoS address information into service registry, so that it does not need a query routing mechanism internally. QoS information of similar or identical services is clustered together. This makes the retrieving and comparison of service QoS very efficient. An autonomous replication mechanism is applied on all peers to adjust load and improve availability.

The rest part of this paper is organized as follows: Section 2 introduces the general model and design consideration of Q-Peer. Section 3 presents how to disseminate QoS and load information in Q-Peer. Section 4 proposes the load balancing approach in Q-Peer. The paper concluded in Section 5 with our future work.

2 System Model

Q-Peer is a peer-to-peer database system for storing QoS information of web services. QoS data is stored in XML documents. Common P2P database has a general requirement that system has to provide an efficient mechanism to query and locate objects, while this requirement can simply be satisfied in Q-Peer by utilizing service registration information. Because no service user cares about service quality without known its function, to query certain QoS metrics without service description is meaningless. Thus, Q-Peer is not an independent P2P database----it has to work with certain service registry system. We organize QoS storage by service description, so that QoS items can simply be located when querying service description. For every service, service registry stores its description and a QoS address list (for replicas). Users retrieve QoS by directly access one of the addresses. In fact, the query mechanism in Q-Peer is similar to the most original P2P system—Napster, by a centralized index server cluster.

QoS can be divided to several classes because same or similar services have same QoS metrics[10]. Functional identical services' quality information is stored at one peer at first, but they could be replicated as a whole when needed. Storing a class of QoS together can improve efficiency because users often retrieve QoS of same service's different implementation to compare and select from them. Different service selection algorithm can be deployed on peers to assist users[1][8]. If a service stores its QoS information at a certain peer, the peer acts as its run-time monitor. Peer updates service QoS periodically. The update process can include certain authentication and evaluation mechanism so that services can not submit fake QoS to Q-Peer.

We do not use super-peer based architecture because super-peer intends to improve query efficiency, which is not a problem in our system. All peers are equal in Q-Peer. Peers employ a replication based load sharing policy which utilizing spare resource on light loaded peers. Every QoS classes can have several replicas on different peers. Service registry has a list of candidate peers for every service and chooses a random one when user request to retrieve QoS. The random peer choosing approach can be substituted with other more sophisticated approach. Every peer has load information about its neighbors for load-balance and backing up each other. The detailed mechanisms will be presented in the following section.

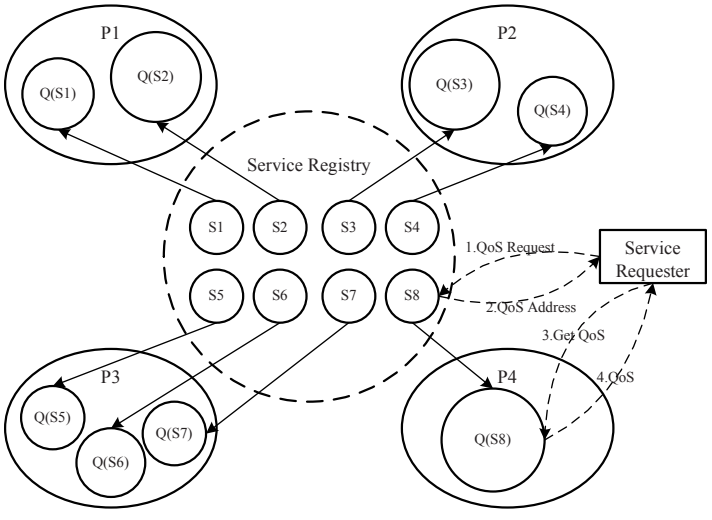

Fig. 1. General model of Q-Peer

Fig.1 illustrates a sample Q-Peer system containing 4 peers and 8 classes of services. Replicas are hided for illustrating our model clearly. Service registry in the figure can be either centralized or decentralized architectures. S_i is a service set which contains a number of same or similar service description. The QoS of a service set S_i is $Q(S_i)$. Each peer stores several sets of $Q(S_i)$. Every service description contains the address of its QoS, like a pointer. When a service requester needs to query QoS of a certain service, it sends a QoS request to service registry, then the registry will reply with a QoS address. Service requester can get QoS by direct accessing the address.

3 Information Dissemination

In Q-Peer, two types of information change frequently which should be constantly updated and properly disseminated in the system. The first is service QoS. The second is load status of peers.

3.1 QoS Update

For a newly registered service *s*, which belongs to service class *S*, it has 2 parts of information to be registered, service description *D(s)* and QoS of the service *Q(s)*. If no service of *S* has been registered before, service registry will choose a random peer to store its QoS information. If *S* has been registered, QoS of the service *Q(s)* is added to the peers storing QoS of the class *Q(S)*. As soon as a peer is informed that it will store a new service's QoS, it contact with the service and get current QoS for the first time.

We have mentioned that for sharing load and improving availability, any *Q(S)* may have several replicas (the replication mechanism is presented in next section). One of the storage peers for a QoS class is the main peer, the others are replication peers. Every time service update its QoS, it update to the main peer first. Then the other replicas are passively updated by the main peer.

3.2 Load Update

In Q-Peer, peer's load and capacity are characterized by the frequency of accessing QoS on a peer. We assume every peer has infinite storage space for cost of increasing storage is much lower than increasing CPU power or network bandwidth. QoS accessing comes from 2 major operations: one is updating of QoS; another is query of QoS. For a peer *P* storing *n* classes of QoS: $\{Q(S_1), Q(S_2), ..., Q(S_n)\}$, each class has an updating frequency f_i^u and a query frequency f_i^q, the load of the peer is:

$$L(P) = \sum_{i=1}^{n} \left(f_i^u + f_i^q \right) \tag{1}$$

A peer *P* has a maxim capacity $C^M(P)$ equals to the estimated maxim allowed accessing frequency $f^M(P)$. The available capacity to accept new service is: $C^A(P) = C^M(P) - L(P)$.

Every peer has a list of other peers' address, called Neighbor List (NL). The neighbor list contains a limited small number of peers which can accept a peer's load sharing request. This list is sorted by C^A in descent order. A neighbor item in NL is $N_i = \langle P_i, C^A(P_i) \rangle, (i = 1...m, a \le m \le b, 0 < a < b)$, where *m* is the total neighbor number, *a* and *b* are the lower and upper limit of *m*. Items in NL can be dynamically added and deleted according to peer status. When a new peer *P* adds to Q-Peer system, it will get a random NL. *P* periodically sends its own $C^A(P)$ to peers in NL and gets their C^A back from reply messages to update its NL. For any peer received an unknown peer's C^A, if it is better than the last item in their NL, the new peer is inserted. If NL exceeds the maxim number limit *b*, the last item will be removed. Peers have a lowest capacity limitation *l* to take a peer as their neighbor. For any N_i which $C^A(P_i) < l$, it will be deleted. If item number in NL is lower than the

minimum number limit, peer will initiate a random walk process to find new satisfied peers. The random walk begins from a random peer in its NL, message containing its own C^A for other peers to update NL if satisfied. For any peer walked through, it sends its C^A back to the initiating peer. The random walk will stop for TTL limitation.

By this load updating approach, peers tend to exchange information with light loaded peers, which is more likely to be able to accept replication requests. For peers having less spare capacity which have not been taken as neighbor of any other peers, they still have chance to use other peers' resource. When they have spare capacity again, they will be added to its neighbor's NL. We have to tune parameters in a more practical environment to limit the message overhead in Q-Peer and improve load sharing.

4 Replication and Load Sharing

If a peer found itself in heavy load, it can ask other peers to replicate some of its service class to share its load. We prefer to replicate service classes as a whole rather than replicate some single service. Because our aim of replication is simply to balance load, to replicate single service could not contribute much to load sharing. And to replicate a part of a QoS class will affect the extended functions like service selection. Thus, a class of QoS is the operation unit of replication.

Every QoS class has $r\left(2 \le r \le K\right)$ replicas including the original one, where *K* is the maxim allowed replica number. To improve availability, the first replica is created immediately after the QoS class is created, so any QoS class has at least 2 replicas. If a peer's load is approaching threshold, it sends replicating request to the neighbor which has the most spared capacity. Peer always tries to replicate the most popular QoS class $Q\left(S_i\right)$. If the spared capacity of the first neighbor can satisfy replication requirement and the class has less than *K* replicas, the first neighbor will be taken as the replication peer. The replication condition is:

$$C^A\left(P_1\right) > f_i^u + \frac{r \times f_i^q}{r+1} \text{ and } r < K \tag{2}$$

In (2), we can find that by replicating a QoS class, replication peer can share $r/r+1$ of the class' query load, but updating load can not be leveraged because all replicas should keep consistency. With the growing of replica number, load sharing by replication can have less and less effect because $r/r+1$ is approaching 1. What's more, keeping more replicas consistent adds more load on the network. Thus the K should be a small number to make the approach effective.

If the spared capacity of first neighbor $C^A\left(P_1\right)$ could not satisfy the replication requirements, the random walk process in previous section will be initiated to rebuild the neighbor list. As soon as a replication peer is found, a replica of $Q\left(S_i\right)$ is transferred to new peer. Service registry is then informed that a new replica can be selected to retrieve QoS.

If all QoS class in a peer has had K replicas and it is still under load pressure, a random QoS class will be chosen to be deleted. Before deletion, service registry is informed so that it will not retrieve the class of QoS from this peer. Main peer of the QoS class is also informed so that it will not update QoS to this peer. If the deleted replica is the main replica of the service class, another replica will be chosen as main replica and related service providers will be informed to update QoS to the new one.

5 Conclusion and Future Works

In this paper, we presented a distributed web service QoS registry—the Q-Peer architecture. The architecture is based on Napster-like unstructured peer-to-peer model. Every QoS item's address is stored in service registry with its service description. Same or similar services' QoS is clustered together to conveniently expand other QoS operation like service selection. Every QoS class has several replicas to improve performance and availability. Replication is based on load status of peers. There is a simple but effective mechanism to exchange load information between peers. Q-Peer architecture is expected to support efficient QoS storage with excellent scalability. It can be used as a QoS infrastructure for global B2B applications.

The design of Q-Peer has just finished and we are currently implementing it. Many detailed design considerations should be tested and adjusted in practical environment. Our future works may include: to design a peer load based replica selection mechanism to help balance load further; to find out a replica deletion algorithm, which will less affect the whole system; to analyze and adjust parameters with experimental results. We will deploy and test Q-Peer on Planet-Lab[9] in near future.

References

1. Liu, Y., Ngu, A.H., Zeng, L.Z.: Qos computation and policing in dynamic web service selection. In: Proceedings of the 13th International Conference on World Wide Web, pp. 66–73. ACM Press, New York (2004)
2. Yu, T., Lin, K.J.: A Broker-based Framework for QoS-Aware Web Service Composition. In: Proceeding of IEEE International Conference on e-Technology, e- Commerce and e-Service (EEE-05), Hong Kong, China (March 2005)
3. Gibelin, N., Makpangou, M.: Efficient and Transparent Web-Services Selection. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 527–532. Springer, Heidelberg (2005)
4. Gnutella Homepage http://www.gnutella.com
5. KaZaA Homepage, http://www.kazaa.com
6. Koloniari, G., Pitoura, E.: Peer-to-peer management of XML data : issues and research challenges. ACM SIGMOD Record, vol. 34(2) (June 2005)
7. Schmidt, C., Parashar, M.: A peer-to-peer approach to Web service discovery. In: Proceedings of the 13th International Conference on World Wide Web, pp. 211–229(2004)
8. Li, F. Su, S., Yang, F.C.: On Distributed Service Selection for QoS Driven Service Composition. In: Proceedings of the 7th International Conference on Electronic Commerce and Web Technologies, EC-Web'06, LNCS, vol. 4082 (2006)
9. Planet-Lab Homepage http://www.planet-lab.org/
10. Maximilien, E.M., Singh, M.P.: A Framework and Ontology for Dynamic Web Services Selection. IEEE Internet Computing 8(5), 84–93 (September 2004)