Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

A QoS Information Dissemination Service for SOA-based CSCW Applications

Xiao Zheng[1,2], Junzhou Luo[1], Jiuxin Cao[1]

[1] School of Computer Science and Engineering
Southeast University
Nanjing, China
{xzheng,jluo,jx.cao}@seu.edu.cn

[2] School of Computer Science
Anhui University of Technology
Maanshan, China

***Abstract*—A fundamental problem that confronts SOA-based CSCW applications is the efficient and timely QoS information obtainment of component services. However, this issue has largely been overlooked. This paper presents a P2P based publish/subscribe service to disseminate new revised QoS information reliably and efficiently. Specialized rendezvous points and a replica mechanism are introduced to reduce the risk of subscriptions loss and consequently improve reliability. A message buffering and packaged delivery mechanism helps to reduce notification traffic. A reverse Chord ring, called RP ring, is designed to quicken subscription delivery and QoS information publication. Node partitioning technique is suggested to tackle the load balancing issue. Simulation results show that the service is reliable, efficient and scalable.**

***Keywords*—Service Oriented Architecture, Computer Supported Cooperative Work, quality of service, information dissemniation**

I. INTRODUCTION

As the cooperative work becomes more and more large and complex, flexibility turns into a great challenge for design of Computer Supported Cooperative Work (CSCW) systems [1]. Service Oriented Architecture (SOA)[2] based applications are built by combining network-available services, which have the features of flexibility, reusability and scalability. Consequently, there has recently been an increase in the use of SOA to build CSCW systems, which typically includes BizTalk[3], CoFrame[4], GATiB-CSCW[5] and so on. In SOA based CSCW systems, basic function blocks and platforms are packaged and published as services, which communicate with each other using a loosely coupled interaction mode.

Quality of Service (QoS) information for services is essential to SOA based applications. Service oriented computing (SOC) environment is usually a dynamic and volatile environment where the QoS parameters of services always change during their lifetime[6]. Due to lack of automatic QoS publication mechanisms, most service providers are unwilling or forget to update QoS information of services after publishing them in Universal Business Registries(UBR). It means that many of the information contained within UBRs is not accurate. Consequently, it consumed more resources when performing Web service binding and took longer to execute communication with unavailable Web services. Accordingly, it is necessary to make service consumers know the new revised QoS information.

Current QoS obtainment mechanisms of services usually involve query-based or monitoring-based methods [7-10]. Query-based methods actively request QoS information of service providers, and usually initiate query periodically or trigger query event under particular conditions. Monitoring-based mechanisms design an engine to monitor QoS in an objective and reliable way. The monitoring engine can run on the provider side, as part of the service middleware, or be placed outside acting as a central proxy. It intercepts the messages exchanged between the service consumer and the provider and outputs an estimate of the delivered QoS.

However, a SOA-based CSCW system is considered as a typical composite service, where many autonomous services are cooperative to perform a task. Current QoS obtainment mechanisms suffer some or all of the following limitations in such environments. Firstly, the service provider usually needs distribute QoS information of services to many interested consumers. This distribution mode is unidirectional one-to-many style. However the request-response mechanism is a one-to-one bidirectional communication style that overloads transport network and not accommodates large-scale QoS information diffusion. Secondly, a service consumer initiates request when it need latest QoS information. This style cannot assure obtaining QoS in time, because the response time is uncertain in a volatile network. In addition, in query-base methods service consumers initiate communication, which is about twice time-consuming than the style of directly distributing messages to consumers. The last problem is scalability. When the monitor intercepts all service invocations, it acts as a central proxy and soon becomes a performance bottleneck.

In order to solve above problems, this paper proposes Pat4C(Pat for CSCW applications), a fully distributed, efficient, fault-tolerant and scalable service for QoS information dissemination based on the publish/subscribe (pub/sub) mechanism. Pat4C aims at reliable and efficient QoS information dissemination for SOA-based CSCW applications.

II. RELATED WORK

Recently, researchers are increasingly turning their attention to building reliable and adaptable CSCW applications based on SOA. In this research field, how to obtain up-to-date knowledge of the current QoS parameters of all the component services is a challenge. In general, QoS parameters can be classified into three categories, based on the approaches to

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

obtain them, which are provider-advertised, consumer-rated and observable parameters[10]. The last two types of QoS parameters are more suitable to be obtained by monitoring-based methods, which are not considered in this paper.

Provider-advertised parameters are usually obtained by query-based methods. Au et al. [7] obtained current parameters of the workflow by querying all the Web service providers when the parameters expire. However, this may lead to some unnecessary queries if the QoS value in fact does not change after its expiration. Improving on this work, Harney et al. [8,9] successively propose two mechanisms called the value of changed information (VOC) and VOC with expiration times(VOC^e) respectively. By VOC or VOC^e, the expected impact of the revised information on the web process could be calculated. If the change brought about by the revised information were worth the cost of obtaining it, query would be preformed. These approaches are selective query. Thus, the mechanisms avoid "unnecessary" queries in comparison with the naive approach of periodically querying all the service providers. However, due to the issues discussed in Section 1, such query-based methods are not suitable for large-scale QoS information dissemination in SOA-based CSCW environments.

Different from above works, we propose a pub/sub system to obtain updated QoS parameters. Pub/sub is an event-based system: clients subscribe to the event service, while servers publish notifications, which will be dispatched to attended recipients[12,14]. The idea of using pub/sub to selective disseminate service information has been used in UDDIv3[11]. However, its pub/sub mechanism is a simple one-to-one distribution style. It could not disseminate large-volume information to multi-consumers. Content-based distributed pub/sub system is a powerful alternative for information dissemination in large-scale distributed networks. Ferry [12] is a well-known content-based pub/sub system, which extensively exploits Chord [13] as the underlying overlay structure to build an efficient and scalable platform to host multiple pub/sub services with unique schemes.

QoS information dissemination services should have such features: scalability, delivery efficiency and reliability. Although Ferry addresses the first two issues, it largely overlooks reliability issues. In addition to solve all the three issues, according to the character of SOA-based CSCW applications, our special service can optimize the performance in many aspects, such as speeding up message delivery and reducing notification traffic.

III. Assumptions and Terminology

This paper only considers provider-advertised QoS parameters, involving cost, response time, capacity, availability etc., which are influenced by provider subjective views[10]. The definitions of these QoS parameters can refer to [6]. Assume that every provider honestly complies with what it advertised, which means the published QoS information is true and worth of trust.

According to the universal content-based pub/sub scheme proposed by [14], a QoS pub/sub scheme could be formally defined as $S=\{U,q_1,q_2,...,q_n\}$, where $U=\{u_i|i=1..n\}$ and u_i denotes the URI of involved service i and q_i denotes a QoS parameter which can be specified by a tuple(QoS expression, type, domain). Typical operators offered by QoS expression are arithmetic and logical operators, such as plus(s_1+s_2), multiplication(s_1*s_2), conjunction(s_1 & s_2), disjunction(s_1 || s_2).The type could be integer, float, string, etc. The domain can be specified as a set of (min,max). The min and max define the range of domain values taken by the given parameter. In order to distinguish the owner of a QoS parameter, a service's URI identifier should be added as a prefix, such as $s_1.Cost$.

The QoS pub/sub scheme is used to produce a QoS subscription or a QoS notification which can be recognized by all members of the pub/sub system. For example, $p=\{(s_1,s_2,s_3), ((s_1.ResponseTime+ s_2.ResponseTime) \| s_3.ResponseTime, 100,+, \infty)\}$ means this QoS subscription p involves three services s_1,s_2 and s_3. The subscription predicate is that the total *ResponseTime* of s_1 and s_2 or the *ResponseTime* of s_3 is more than 100. A QoS notification is the URI of a Web service with a set of QoS parameters $\in S$ and it can be represented as $e=\{ u=u_1,q_1=c_1,q_2=c_2,\cdots, q_n=c_n\}$. u_1 is always a const string which denotes the definite URI of a Web service. A predicate c_i has a name, type, min, max and probability and is used to specify a constant value or range with probability for a parameter. It can be simplified as (*min*, *max*, *probability*). For example, [*ResponseTime*, float, <(0,0.2,40%), (0.2, +∞,60%)>] means QoS parameter *ResponseTime* is less than 0.2s with probability 40%, and more than 0.2s with probability 60%.

IV. System Design

Pat4C is essentially a rendezvous network[16] built on top of Chord to support QoS information distribution. Figure 1 illustrates its architecture. It consists of two kinds of components, client and rendezvous point(RP). For example, r_i (i=1..4) are RP nodes and c_i (i=1..5) are clients in figure 1. Clients can serve as service providers or service consumers (QoS subscribers). RP is a meeting point for advertisements and subscriptions, which is known to both providers and consumers. Different from most of existing pub/sub systems, Pat4C's RP node is a specialized node set by system and has an invariable and fixed ID. The idea is that the system needs reliable RP nodes and should not make consumers or providers process heavy job including subscription management and matching.

Given a QoS pub/sub scheme $S=\{U,q_1,q_2,...,q_n\}$, the node ID of an RP node equals to $k_i=h(q_i)$, where k_i is a key derived from an parameter q_i by using the consistent hash function $h()$, which is used in Chord to produce node IDs and data keys. The total number of RP nodes in a system is therefore equal to the number of parameters in the unique scheme. RP nodes are interconnected to form a reverse Chord ring, called RP (Rendezvous Point) ring. There are therefore two rings in the system. One is a Chord ring consisting of RP nodes and clients, and another is a RP ring only involving RP nodes. Each node contains a Chord finger table and a Chord successor list, which are as same as those used in Chord. RP node need additionally maintain a RP ring's finger table and successor list. In Chord, the successor list and finger table record routing information, which consulted by the host node to route a message with a key k to a destination node whose ID is the successor of k.

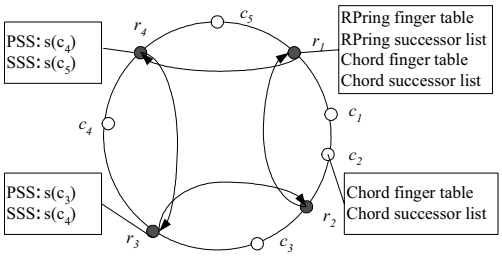

Figure 1. Structure of Pat4C. s(x) denotes all subscriptions from client x.

A. Subscription installation and management

When a consumer wishes to subscribe for some QoS information, it has to publish its interests to a RP node in the form of subscription *s*=(*sid*, *p*). *sid* denotes node ID of the subscriber, *p* is its QoS subscription. Algorithm 1 outlines the installation process, called InstallAroundRP. The basic idea behind InstallAroundRP is that a subscription *s* is stored in two RP nodes around *s*'s *sid*. Figure 1 illustrates the result of installation. Subscriptions from c_4 will be delivered to r_4 through Chord ring, and subsequently be delivered to r_3 through RP ring. Therefore, r_3 owns subscriptions from c_3 and c_4 which belong to two regions: one region precedes r_3, another follows r_3. In a RP node, all subscriptions produced by previous client nodes on the Chord ring are called predecessor subscriptions which compose a Predecessor Subscription Set(PSS) , and those from following clients are called successor subscriptions which compose a Successor Subscription Set(SSS). Predecessor subscriptions are replicas that are used to recover subscriptions after their host RP node failures. We only need to deal with the management of successor subscriptions. In this mode, a subscription only traverses a fraction of the Chord ring and doesn't need to traverse the whole ring. Similarly, a QoS notification also needs to be delivered among this fraction only. Compared to randomly choosing a RP node to install subscriptions or selecting the previous RP node suggested in[12], our algorithm avoids sending the redundant messages across the Chord ring space and making the message traverse a shorter path of the system.

In order to use Chord's routing approach to delivery QoS notification, a subscription table is built to map subscriptions to neighbor nodes including successor nodes and finger nodes. As illustrated in figure 2, the entry of a subscription table is composed of *neighbor* and *subscription*, where *neighbor* denotes a successor node or finger node and *subscription* denotes subscriptions in SSS. A subscription *s* is stored in the entry of a neighbor whose node ID is equal to or most immediately precedes *s.sid*. For example, subscriptions from c_{i2}, c_{i3} are stored in the entry of f_{i1} because their ID is in the range from f_{i1} to f_{i2}. Note that successor nodes and finger nodes are also clients in Pat4C. In order to show their features of routing tag, they are labeled by s_{ij} (j=1,2) and f_{ij} (j=1..3) respectively in figure 2.

Algorithm 2 outlines how to store a subscription. Because PSS is used for subscription backup, its subscriptions do not need be considered. A RP node only manages SSS.

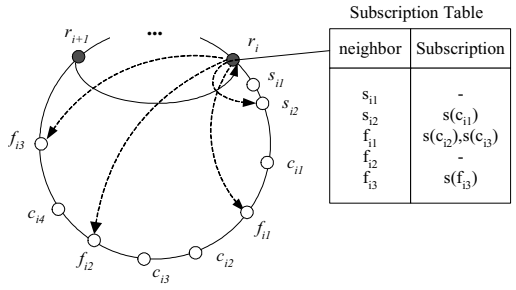

Figure 2. A subscription table.s(x) denotes all subscriptions from client x.

Algorithm 1 InstallAroundRP(Subscription s)
1: choose a RP node r_1 which is the most immediate successor of s.sid among all RP nodes
2: store s in r_1
3: store s in another RP node r_2 which is the direct successor of r_1 in the RP ring
Algorithm 2 StoreSubscription(Subscription s)
1: find the neighbor node n_i whose ID is equal to or most immediately precedes s.sid
2: store s into n_i's entry
Algorithm 3 PublishQoSInfo(QoSInformation e)
1: find a nearest RP node r_1 and send e to it
2: n = the index of hash space 2^n
3: BinaryDeliveryinRPring(e, r_1, n) //send e to all RP nodes in the RP ring
Algorithm 4 BinaryDeliveryinRPring(QoSInformation e, RPnode r, Round n)
1: select a RP node r_m which is the most immediate successor of r's ID+ 2^{n-1} in r's reverse finger table
2:**if** r_m!= r **then**
3: send e to r_m
4: BinaryDeliveryinRPring (e,r,n-1)
5: BinaryDeliveryinRPring (e,r_m,n-1)
6: **endif**

Figure 3. Algorithms used in Pat4C

B. QoS information publication

Because any RP node may have same subscriptions, QoS information will be published to all RP nodes. RP ring could quicken QoS information delivery among all RP nodes. The publication algorithm outlined in algorithm 3 also utilizes underlying Chord routing protocol to transfer QoS notification. Firstly, service providers find the nearest RP node among all RP nodes and send a new QoS notification message to it. Secondly, the message is distributed to all RP nodes of the RP ring through a recursive process. Note that algorithm 3 and 4 outline the process of publication from a whole view. In fact all RP nodes in the RP ring will perform this publication operation in parallel.

C. QoS notification buffering and packaged delivery

QoS notification messages only need to traverse a fraction of the Chord ring space due to the fact that each RP node only stores those subscriptions from a nonoverlapped, contiguous region of the Chord ring space. According to algorithm 1, a subscription is stored in two different nearest-neighbor RP nodes. Each RP node therefore stores QoS subscriptions coming from its two sides. Each RP node only need deliver

QoS notification to its subsequent matched subscribers along clockwise ring in order to get shorter delivery path.

There are plenty of notification messages in pub/sub systems. It is important to reduce such messages because any reduction in the number of messages leads to a reduction in the overhead involved in packaging and delivering each individual message, and to an improvement in scalability. SOA based applications are usually implemented by interaction and collaboration of multiply services that construct a composite service according to a definite business workflow. If it is only allowed to subscribe to component services within a composite service respectively, the subscriber or service consumer will hardly be satisfied with plenty of invaluable notifications. For example, figure 4 shows a composite service described by WSC graph we suggested in [16]. s_1,s_2 and s_3 compose a parallel execution path, which means s_1 and s_2 must be executed in parallel with s_3. The consumer of the composite service only considers the end-to-end QoS from s_s to s_d. No matter how the QoS of s_1 or s_2 changes, it is good that the whole QoS is smaller than a threshold. It is possible that the QoS of s_1 increases while that of s_2 decreases, and the total QoS of s_1 and s_2 does not change. Only considering the subscription of a single component service is senseless for a composite service. The QoS of a composite service is decided by component service in it. References [6,17] introduce how to calculate the QoS value of a composite service according to that of the component services.

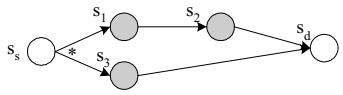

Figure 4. Example of a composite service

Due to a composite service consisting of several component services whose QoS notifications reach RP nodes at different time, a buffer should be set at each RP node to temporarily store these incoming notifications. When all involved component services in a certain subscription can be got in a buffer, the end-to-end QoS will be calculated to compare with the subscription. If the matching succeeds, all the related QoS information will be packaged to a single notification. Because all the subscriptions of composite service are stored in a same RP node, end-to-end QoS can be calculated locally.

The method of end-to-end QoS subscription reduces the notification traffic in two aspects. An experiment described in Section 5 shows the quantity of the reduction. Firstly, upon QoS information of component services stored in the buffer is matched with an end-to-end QoS subscription, the RP node will create a notification and deliver it to interested subscribers. Thus this mechanism reduces traffic generated by messages of component services. Secondly, this method also can reduce small and frequent notifications. After using this method, multiple QoS information of component services can be packaged within a single notification message. This way, the headers that would have been transmitted with every individual message are reduced to a single header on a grouped message. In addition, an end-to-end QoS criterion is calculated at the RP node, which helps to reduce subscribers' overhead and improve their efficiency.

In Ferry, events are disseminated along an $EmdTree_r$ [12] from the RP node r to the subscribers. $EmdTree_r$ is an embedded tree rooted at node r which is formed by the DHT overlay links. Pat4C also utilizes $EmdTree_r$ to disseminate QoS notification. For the sake of space limitation, we will not discuss them in detail.

D. Fault-tolerance

Node failure is a common phenomenon in a volatile network environment. Throughout the entire design, fault-tolerance plays an important role in a reliable QoS dissemination system. Our service takes advantage of the fault-tolerance and recovery mechanisms provided by the Chord routing layer. This enables it to survive multiple link and RP node failures and adapt its routing state so that it can still deliver QoS notifications to subscribers.

Client nodes include service providers and service customers. Service customers subscribe to QoS information. Their failures do not influence the whole pub/sub system. But service providers' failures mean that they stop providing corresponding services. This situation must be notified to interested consumers, however, which could not be actively advertised by the failure node. Pat4C sends a failure notification to all RP nodes after Chord ring having been repaired.

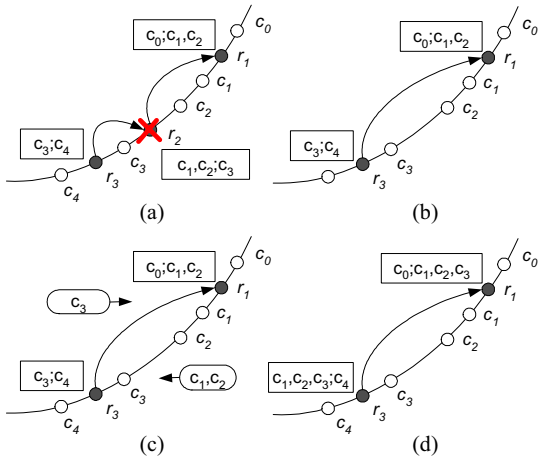

Figure 5. Recovery after RP node r_2 fails

A RP node is a key point of the pub/sub system. If a RP node fails, all subscriptions stored at it will be lost, which will lead to corresponding subscribers not receiving new QoS notifications any more. Pat4C uses two methods to avoid RP node failure and recover subscriptions once a RP node indeed fails. Firstly, all RP nodes in Pat4C are specified and high reliable servers can serve as these nodes, which can be done in a real CSCW system. Secondly, algorithm 1 replicates subscriptions to another neighbor RP node. This guarantees not losing subscription to a large extent. Once a RP node fails, Pat4C will immediately recover the RP ring and replicate subscriptions between the new linked RP nodes. Figure 5 illustrates such process. (a) r_2 fails and subscriptions from c_1,c_2 and c_3 are also lost. (b) Pat4C recovers RP ring and Chord ring through underlying Chord algorithm. (c) r_1 delivers

subscriptions of c_1 and c_2 along Chord ring, and r_3 send subscriptions of c_3 to r_1 by one hop. (d) Subscriptions are recovered. The nodes IDs preceding a semicolon in figure 5 indicate predecessor subscriptions, and ones following the semicolon indicate successor subscriptions.

E. RP node partitioning

In Pat4C, the number of RP nodes is equal to the number of parameters in the QoS pub/sub scheme. This limitation also significantly reduces the scalability of the whole system. If the number of parameters is small, a large number of subscriptions and QoS information may overload the RP nodes. Moreover, it is possible that some RP nodes have plenty of subscriptions and others have few.

Subscription partitioning [18] is an effective load balancing technical that divides scheme's attribute range into several sub-ranges. Pat4C adopts this technique to tackle the load balancing issue. For example, consider that a QoS pub/sub scheme *S* has a parameter *Cost* and its value range is [0,100]. Without partitioning, there is only one RP node. If we partition *Cost* into several nonoverlapped contiguous ranges, [0,30],(30,60) and (60,100), we may create three RP nodes by hashing the parameter name *Cost* with a range. Thus, this method can produce more RP nodes and distribute load over them. Note that, once a RP node is overloaded, a partitioning action will be taken.

V. SIMULATION RESULTS

This section evaluates our system by simulation. Our simulation is based on p2psim which is a widely-used multi-threaded, discrete event simulator to evaluate and investigate P2P based systems. The QoS pub/sub scheme *S* used in our experiments is defined as

S={[URI,string],[Cost,float,< 0,+ ∞ >], [ServiceIsAlive, boolean], [ResponseTime, float,< 0, +∞>], [Capacity, integer, <1, 100>], [Availability, float,<0, 100>]}.

The simulations were initialized with a Chord ring consisting of five RP nodes because of only five parameters in *S*. Subsequently a reverse RP ring was also created. A new client node joins the system at a randomly chosen time until the total number of nodes reaches a bound. When the system reached a stable state, we scheduled subscription installation events into the system to store the subscriptions. After QoS subscription installation, the QoS information publication was modeled as exponential distribution with an average interarrival time of 60 seconds. The QoS notification was generated randomly from *S* and we used 5,000 QoS information in simulations.

Four metrics are listed below to evaluate the performance and cost of Pat4C. **Hops** are the average number of overlay hops taken by a pub/sub system to deliver a message to all of its subscribers. **Delivery overhead** is the ratio of the number of intermediate nodes involved during the delivery of messages to the number of interested subscribers. **Load imbalance ratio** is the ratio of maximum load to the minimum load among all RP nodes. The lower the load imbalance ratio, the better the performance of load balancing. **Notification traffic** is the average number of notification messages.

Firstly, we examined the performance of Pat4C. Figure 6 and 7 show hops and overhead with different network sizes and various percentages of nodes as subscribers respectively. As the network size increases from 1,000 to 10,000, the hops and overhead incurred by QoS notification delivery increase modestly. This shows that Pat4C can scale to a large number of nodes. Figure 6 also shows that the percentages of nodes as subscribers hardly impact hops. With the percentages of nodes as subscribers for the same QoS notification increasing, the overhead decreases significantly. This is because messages transfer among more subscribers upon more intermediate nodes becoming subscribers. Hence, Pat4C is very suitable for delivering messages to a large number of subscribers.

Table 1 shows that subscription distribution changes from imbalance to balance after RP node partitioning. Initially, there are 5 RP nodes, and the load imbalance ratio is 2.5. After subscription partitioning the load balance ratio decreases rapidly and reaches 1.2 upon 5 new RP nodes being created. This is because new RP nodes averagely bear some load from the RP node of higher load.

Following experiments were conducted to examine our optimization for reduction of notification traffic. The composite service described in figure 4 serves as a model to generate subscriptions for component services and composite services respectively. The generated end-to-end subscription and the subscriptions for s_1, s_2 and s_3 have the same goal. QoS information of s_1, s_2 and s_3 are generated at random, the number of which is from 100 to 1000. Figure 8 shows the comparison in the number of matched and generated notification messages, which will be delivered to the subscriber.

Pat4C and Ferry have different design purposes. Pat4C is specially used for QoS notification dissemination, while Ferry is a universal platform for multiple pub/sub services. However, because they all use Chord as underlying links, we can compare them in terms of message delivery performance. Pat4C adopts an event delivery algorithm similar to Ferry's, so we compare Pat4C and Ferry in terms of subscription installation. Figure 9 shows the hops of Pat4C and Ferry in different network sizes. Compared with Ferry, Pat4C can dramatically reduce hops. This is because the RP ring helps quicken messages delivery. Ferry's PredRP algorithm must install subscriptions at subscriber's preceding RP node which will lead to messages traversing along the whole Chord ring space, while Pat4C's InstallAroundRP algorithm not only quickly delivers subscriptions to the preceding RP node through reverse one hop of RP ring, but also creates a replica at subscriber's successor RP node.

TABLE I. SUBSCRIPTION DISTRIBUTION WITH RP NODE PARTITIONING

Number of RP nodes	**5**	**6**	**7**	**8**	**9**	**10**
Load imbalance ratio	2.5	2.3	1.9	1.5	1.3	1.2

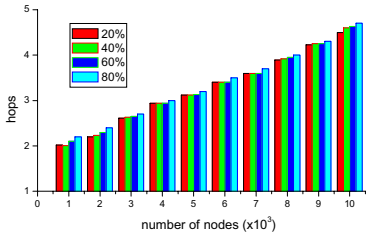

Figure 6. Comparison of hops with different network sizes and various percentages of nodes as subscribers.

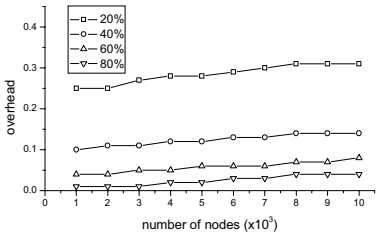

Figure 7. Comparison of overhead with different network sizes and various percentages of nodes as subscribers.

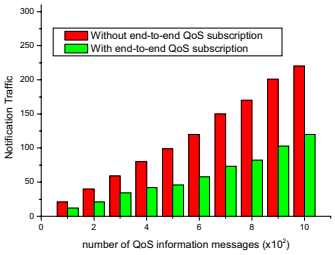

Figure 8. Comparison of notification traffic with/without end-to-end QoS subscription.

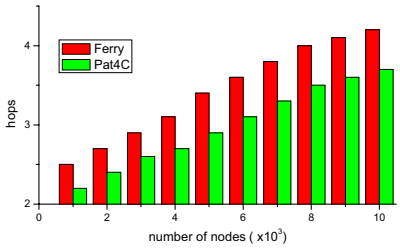

Figure 9. Comparison between Pat4C and Ferry in subscription installation.

VI. CONCLUSIONS

Pat4C is a P2P based pub/sub service for QoS information dissemination of Web services. It not only maintains the scalability of the underlying Chord network, but also improves reliability and timelines guarantee especially for QoS information obtainment. Compared to query-based and monitoring-based mechanisms, Pat4C has the advantages of low cost, timely distribution and scalability, which is suitable to large-volume QoS information distribution. Moreover, Pat4C, built into SOA based CSCW applications, could be used to provide updated QoS information of services for maintaining the flexibility of the whole applications.

Though our work has achieved initial success, some issues need to be explored in near future. For example, if two adjacent RP nodes fail, the collective subscriptions stored in them will be lost and never be recovered. The possibility of such situation happening will be evaluated and a more reliable method will be built into Pat4C in near future.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China under Grants No. 60773103, China Specialized Research Fund for the Doctoral Program of Higher Education under Grant No. 200802860031, Jiangsu Provincial Natural Science Foundation of China under Grants No. BK2007708 and BK2008030, Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No. BM2003201 and Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education under Grants No. 93K-9.

REFERENCES

[1] P. H. Carstensen, K. Schmidt," Computer Supported Cooperative Work: New Challenges to Systems Design," Kenji Itoh (ed.). Handbook of Human Factors, Tokyo, 1999.

[2] M. N. Huhns, M. P. Singh,"Service-Oriented Computing:Key Concepts and Principles," IEEE Internet Computing, 2005, 9(1): 75-81.

[3] Learn BizTalk Sever. http://msdn.microsoft.com/zh-cn/biztalk/aa937649(en-us).aspx

[4] J. Jiang, S. Zhang, Y. Li, M. Shi, "CoFrame: A Framework for CSCW Applications Based on Grid and Web Services," IEEE Int'l Conf. on Web Services (ICWS'05).2005.

[5] K. Stark, J. Schulte, T. Hampe,"GATiB-CSCW, Medical Research Supported by a Service-Oriented Collaborative System," CAiSE 2008, LNCS 5074, pp. 148–162, 2008.

[6] S. Ran, "A Model for Web Services Discovery with QoS", ACM SIGecom Exchanges, 2004,4(1),1-10.

[7] T.-C. Au, U. Kuter, D. S. Nau, "Web Service Composition with Volatile Information", International Semantic Web Conference, 2005, pp. 52–66.

[8] J. Harney and P. Doshi, "Adaptive Web Processes Using Value of Changed Information", Proc. Int'l Conf. on Service-Oriented Computing,2006, pp. 179–190.

[9] J. Harney and P. Doshi, "Speeding up Adaptation of Web Service Compositions Using Expiration Times", Int'l World Wide Web Conf., 2007, pp.1023-1032.

[10] L. Zeng, H. Lei, and H. Chang, "Monitoring the QoS for Web Services", Proc. Int'l Conference on Service-Oriented Computing, LNCS 4749, 2007, pp. 132–144.

[11] L. Clement, A. Hately, C.V. Riegen, T. Rogers, "Universal Description Discovery & Integration (UDDI) 3.0.2. 2004", http://uddi.org/pubs/uddi_v3.htm

[12] Y. Zhu and Y. Hu,"Ferry: A P2P-Based Architecture for Content-Based Publish/Subscribe Services", IEEE Trans. On Parallel and Distributed Systems, 2007, 18(5),672-685.

[13] I. Stoica, R. Morris, et al.,"Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications", Proc. ACM SIGCOMM, Augest 2001, pp. 149-160.

[14] F. Fabret, H.A. Jacobsen, et al., "Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems", Proc. ACM SIGMOD, 2001, pp. 115-126.

[15] I. Stoica, et al., "Internet Indirection Infrastructure", Proc. ACM SIGCOMM, Pittsburgh, PA, 2002, pp. 73-86.

[16] X. Zheng, J. Luo, et al., "Ant Colony System Based Algorithm for QoS-Aware Web Service Selection", Int'l Conf. on Grid Service Engineering and Management (GSEM 2007), September 2007, LNI 117, pp.39-50.

[17] D. A. Menascé, "Composing Web Services: A QoS View", IEEE Internet computing, 2004,8(9), pp.88-90.

[18] Y.-M. Wang, L. Qiu, D. Achlioptas, et al., "Subscription Partitioning and Routing in Content-Based Publish/Subscribe Systems," Proc. 16th Int'l Symp. Distributed Computing (DISC '02), Oct. 2002.