International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

A Scalable, Hybrid Peer-To-Peer Directory Service for Mobile Devices

Kailash Prasad Dewangan[1], Lalita Nayak[2]

[1]Department of Computer Science and Engineering, Kruti Institute of Technology and Engineering, Raipur(C.G.), India
kaishapt@yahoo.com

[2]Department of Computer Science and Engineering, Kruti Institute of Technology and Engineering, Raipur(C.G.), India
lalitanayak2010@gmail.com

Abstract: ***As we know that, mobile devices could and should be enabled to provide server functionalities. Co-ordination among applications requires a directory service that can provide a variety of functions. It brings the challenge to solve the scalability and dynamism for traditional directory service. In order to improve scalability, we develop a peer-to-peer based directory service that is built on distributed hash tables. We designed an adaptive load-balancing scheme to reduce hotspots and distribute the load among the directory servers according to their load and capabilities. Distributed hash tables (DHTs) provide guarantees of an upper bound on the number of messages to find a key. In order to handle dynamism in the information, we develop a push interface in the directory servers besides the traditional pull interface. The push interface provides a mechanism to reduce the query load in the server by pushing information to the clients when the clients need real-time updates for resources.***

Keywords: Peer Directory Service, Resource Hash Table, Distributed Hash Table, Peer Virtualization.

1. Introduction

PeerDS is a peer directory server that provides publishing, resource and group management to resource providers, and provides pull and push interfaces to clients. A resource hash table (RHT) is composed of keys, PeerDS nodes and summary of properties of the keys. A key is a hashed value of the name of a resource object, its group and its category.

Properties of the key provide the functional and/or non-functional description of the service, resource or group associated with the key. The routing table (DHT) in each PeerDS node keeps track of a subset of all PeerDS nodes. The DHTs provide the routing among PeerDSs. All PeerDSs form a PeerDS ring as depicted in Figure 1.

A PeerDS node has a routing table, a successor node set and the predecessor for each peer identifier associated with the node. The node also stores part of the global directory database. In a PeerDS node, there are four interfaces: a publish interface that provides publish functionality for resource providers; a pull interface that provides regular lookup operations for directory clients; a push interface that provides subscription services to directory clients; and a peer network interface that supports communication among PeerDS nodes such as routing.

2. Hybrid Interface

The hybrid interface of directory service is described in Figure 2. A pull interface is typically provided in the regular directory services. Usually a client sends a lookup request to the directory server. After the directory server looks up the directory database, it sends the result back to the client.

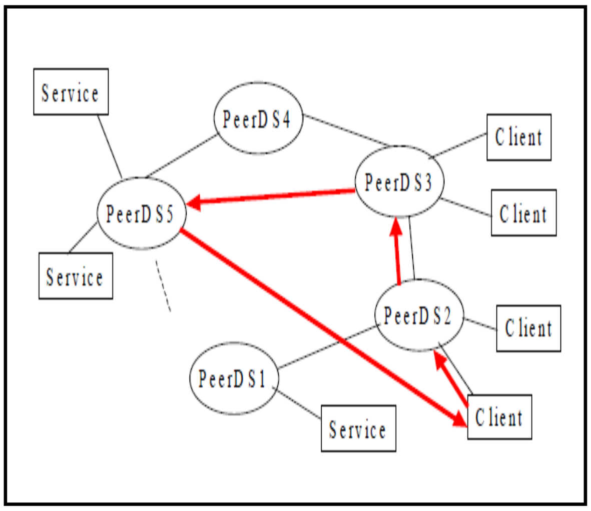

Figure 1: PeerDS system Architecture

When a client wants to know the real-time updates of a resource such as location, it will frequently send requests to the directory server to avoid missing some important updates. If millions of clients choose to do this, as this is often the case in some important events such as Super Bowl games, the directory server will be easily overloaded.

Based on this observation, we could provide a push interface to push this information to the interested clients. A client can subscribe to the information it is interested in and specify a filter function so that only useful information would be transmitted back to client. Since many clients may be interested in the same information, the directory server only needs to process once for these groups of clients.

In this way, we could improve the scalability of the directory server and reduce the communication overhead between a directory server and its clients. If there are many mobile clients, we could also move some computing functions to the directory server to save the energy in the clients. For example, rather than the client receiving the number of bullets remaining with each soldier and then totaling the number, the directory server could summarize them and send only the sum to the client.

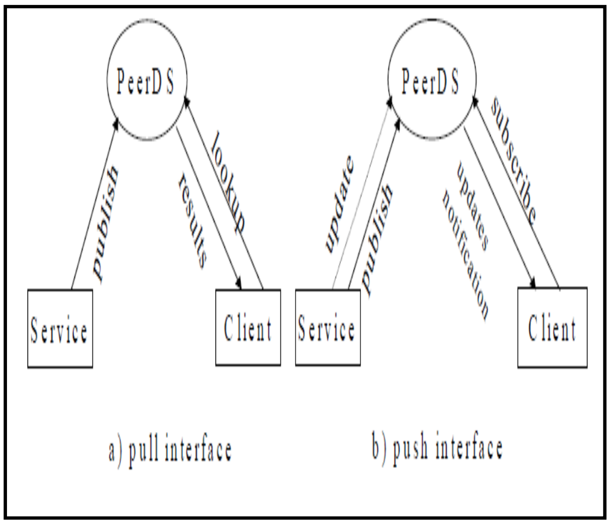

Figure 2: Hybrid Interface

Figure 3 describes the scalability comparison between the push interface and the pull interface. The scalability is measured by considering the changes of load at the server with number of requests (clients) in both push and pull interfaces.

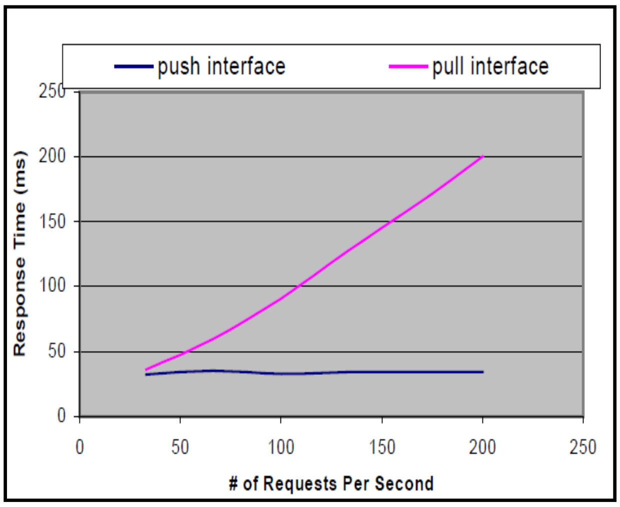

Figure 3: Scalability comparison between Push Interface and Pull Interface

The server load was measured by the response time perceived by the directory clients. Each lookup request to the pull interface requires a database operation that needs around 20ms. With the increase in the number of requests per second to the directory server, the server load in the push interface keeps almost constant since the server only needs one database operation for all subscribers to the same channel. However, the server load increases dramatically in the pull interface operations.

3. System Model

Structured P2P systems such as DHT based P2P systems provide an upper bound on the number of messages so that they guarantee the answer if the result is in the P2P network. As we can see from Chord, CAN, Pastry, Tapestry, PeerCQ and Catalog, this feature is based on the design of identifiers in the distributed hash tables. There are two identifiers in a virtualized P2P system: peer identifier and resource identifier.

In this paper, we mainly focus on peer identifier and resource identifier. In order to map a resource identifier to a peer identifier, both identifiers are carefully designed in an m-bit identifier ring modulo 2m, where m is a system parameter (m=24 in our study) and 2m is the identifier space, so that a peer node can be identified when a resource identifier is known.

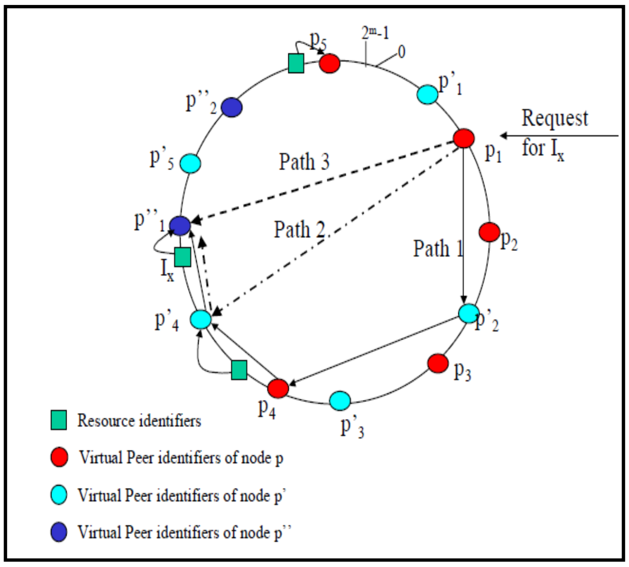
p_5
2^m-1
0
p'1
Request for I_x
p1
p''2
p'5
Path 3
Path 2
Path 1
p''1
p2
I_x
p'4
p'2
p3
p4
p'3
Resource identifiers
Virtual Peer identifiers of node p
Virtual Peer identifiers of node p'
Virtual Peer identifiers of node p''

Figure 4: Identifier Ring

The identifier ring is depicted in Figure 4. A physical node could be associated with multiple virtual peer identifiers. P, P'', P''' are three physical nodes. Virtual peers P1, P2, P3, P4 and P5 are located in node P. Virtual peers P1', P2', P3', P4' and P5' are located in node P'. Virtual peers P1'' and P2'' are located in node P''.

There are two layers of P2P networks in our system. One is the virtualized P2P network that we use to publish / lookup resource objects. The other is the physical network on which we maintain load balancing. Our top k peer selection algorithm is executed in the physical P2P network that has a much smaller number of peers than the virtualized P2P network. We use the new routing algorithm in the virtualized P2P network and Chord protocol in the physical one.

3.1 Peer Virtualization Data Model

Let Ip denote a peer identifier, Ir denote resource identifier, Np denote the number of peer identifiers, Nr denote the number of resource identifiers. Usually Nr >> Np.
Properties: Properties(Ip) of a peer identifier Ip describe the address, port, capabilities, node class and load information.

Predecessor: Predecessor(Ip) of a peer identifier Ip is the maximum peer identifier that is less than Ip in the peer identifier ring.

Successor Node Set SuccessorSet (Ip) of a virtual peer identifier Ip is defined as:
SuccessorSet (Ip) = { Ipi | Ipi = SUCC_NODE (Ip, i) (0<=i<r)}
in which SUCC_NODE (x, i) returns the ith minimum peer identifier which satisfies two conditions: i) it is greater than x in the peer identifier ring and ii) the node associated with this peer identifier x is different from the nodes that are already in the successor node set, and r is the number of successive nodes maintained.

Routing Table RoutingTable (Ip) of a peer identifier Ip for node p is defined as: RoutingTable (Ip) = { (Ipi, Address(Ipi)) | Ipi = MIN_NODE ((Ip+2i-1) mod 2m) (0<=i<=Np)} in which MIN_NODE (x) returns the minimum peer identifier which is greater than or equal to x in the peer identifier ring, and Address (x) returns the physical IP address of the peer identifier x.

Peer Identifier Descriptor: PeerDescriptor(Ip) of a peer identifier can be defined as:
{Ip, Properties (Ip), Predecessor (Ip), SuccessorSet (Ip), RoutingTable (Ip)}

Node Descriptor NodeDescriptor(p) of a node p is defined as {(Ipi, PeerDescriptor(Ipi)) | Ipi is one of virtual peers residing in node p.} The comparisons in the above definition assume modulo 2m operations. In the following discussion, all comparisons assume modulo 2m operations unless otherwise specified. If peer p' is said to be closer to peer p than p'' is close to p, that means p' is in the clockwise path from p'' to p' in the identifier ring. (This is a very important point to understand the algorithms.) The routing node table is used for routing the information among virtual peer nodes. The successor node set is used for load balancing and fault tolerance. As we will discuss in the routing algorithm, the successor node set could also be utilized to speed up routing.

4. Algorithms

4.1 Routing Algorithm

In previous DHT protocols such as Chord, only the routing table of the peer identifier is used for the routing protocol. In our system, we utilize both the routing node table and successor node set of *all* peer identifiers in the node for the routing of a message.

As we previously discussed, we assign varying numbers of virtual peers to a node according to the capabilities and the load of the node. A node p is associated with the node descriptor NodeDescriptor(p). The idea to speed up the routing process is to utilize the shared information in the node descriptor as the computation in the local node is much cheaper than the message communication among nodes.

A typical situation of routing is to locate the proper peer identifier Ip given a resource identifier Ir. The algorithm to find the next peer identifier Ip' to which the request is forwarded from the current node p is described in Table 1. FIND_CLOSEST_NODE () returns the peer identifier, which is the clockwise closest peer in the identifier ring to the destination peer.

Table 1: Finding the Next Peer Identifier (Routing Algorithm)

```
0 Routing (Ir, Ip, p) {
1 If (Ir == Ip) return (Ip, p); // find the peer identifier and the node
2 Else {
3 (Ip', p') = (Ip, p) // initialize
4 For (i=0; i < GetNumberOfPeerIDs(p); i ++) {
5 // Find the closet peer identifier (Ip'', p'') in the routing table of this peer identifier
6 (Ip'', p'') = FIND_CLOSEST_NODE (Ir, p.PeerIDs(i).RoutingTable);
7 // Find the closet peer identifier (Ip'', p'') in the successor node set
8 // of this peer identifier
9 (Ip''', p''') =FIND_ CLOSEST_NODE(Ir, p.PeerIDs(i).SuccessorSet);
10 // Compare (Ip', p') with (Ip'', p'') and (Ip''', p''') to find the closet
11 // peer identifier in these three identifier pairs.
12 // Assign the closet peer identifier and its node to (Ip', p');
13 (Ip', p') = FIND_ CLOSEST_NODE (Ir, {(Ip', p'), (Ip'', p''), (Ip''',p''')});
14 }
15 return (Ip', p');
16 }
17 }
```

Throughout this chapter, the old routing algorithm refers to Chord routing algorithm in the virtualized P2P network. The new routing algorithm refers to our routing algorithm. The key difference between this routing algorithm and the old one is the loop from Line 4 to Line 14. The old routing algorithm will not search the routing tables and successor node sets of *all* virtual peer servers residing in the same node. This routing algorithm is not limited to our peer virtualization scheme. If a peer virtualization scheme does not maintain a successor node set for each virtual peer, the routing algorithm could ignore Line 9.

Now we use an example to compare the new routing algorithm and the old routing algorithm. In Figure 2, P, P' and p'' are three physical nodes. Each physical node may be allocated multiple virtual peers. P1, P2, P3, P4 and P5 are virtual peers residing at the same node P. P1', P2', P3', P4' and P5' are virtual peers residing at the same node P'. P''1 and P''2 are virtual peers residing at the same node P''. A query request for resource Ix is made to the virtual peer P1.
PATH 1 (P1->P'2->P4->P'4->P''1) shows the routing path in the old routing algorithm, which only searches the routing table of the virtual peer.

PATH 2 (P1-> P'4->P''1) shows the routing path if we search all the routing tables of all virtual peers residing at the same node. PATH 3 (P1-> P''1) shows the routing path of the new routing algorithm, which searches all the routing tables and successive Node Set of all virtual peer servers residing in the same node. As we can see from Figure 14, PATH 1 needs 4 hops to reach the destination, PATH 2 needs 2 hops to reach the destination and PATH 3 only needs one hop to reach the destination.

Hypothesis 1: Given a resource identifier Ir, the routing in the virtual peer pp', which is closer to the destination virtual peer pp in the clockwise direction at the identifier ring than the other virtual peer pp'', takes equal or less number of messages to reach pp than the routing in a peer pp''. We formalize the hypothesis as follows:

Let PATH (pp', pp, Ir) be the routing path from p' to p and PATH (pp'', pp, Ir) be the routing path from pp'' to pp. Let Distance (pp', pp, Ir) be the number of hops of PATH(pp', pp, Ir) and Distance(pp'', pp, Ir) be the number of hops of PATH(pp'', pp, Ir).

If pp' is closer to pp than pp'' is close to pp in the clockwise direction, we have
Distance (pp', pp, Ir) <= Distance (pp'', pp, Ir)

The hypothesis is correct if pp' and pp'' are virtual peers that reside in the same physical node pp* since our routing algorithm will search routing tables and successor sets in pp* to find the same or closer next peer so that Distance (pp', pp, Ir) <= Distance (pp'', pp, Ir).

Table 2: Lookup Operation

```
0 Lookup (resource_category, resource_group, resource_object_name, resource_properties) {
1 // Generate the resource identifier based on the resource group
2 // and resource object name
3 Ir= GenerateResourceIdentifier (resource_category, resource_group,
4 resource_object_name);
5 (Ip', p') = Routing (Ir, Ip, p);
6 (Ip'', p'') = (Ip, p);
7 // Find the node that stores the information about the resource object.
8 While ((Ip', p') != (Ip, p)) {
9 Forward the lookup requests to node p'
10 Continue the lookup operation in node p'
11 (Ip",p") = (Ip', p');
12 (Ip', p') = Routing (Ir, Ip', p');
13 }
14 Now the node p' is the node that stores the information about the resource object.
15 Query the resource database in the node p' to return the records that satisfies resource properties including both functional properties and non-functional properties.
16 }
```

5. Conclusion

Structured peer-to-peer systems are popular solutions for large scale distributed computing and query processing. We implement a scalable peer-to-peer based directory service called PeerDS, which is built on an improved distributed hashed table protocol. PeerDS supports both pull-based queries and push-based update multicasts to address dynamism, heterogeneity, complexity and scalability of information.

Heterogeneity among peers calls for peer virtualization to maintain a simple, yet powerful peer-to-peer overlay network. Nevertheless, peer virtualization generates a huge number of virtual peers and causes the unnecessary communication overhead in the routing process. In this paper, we propose a new peer-to-peer routing algorithm that reduces the number of hops of message forwarding and improves the performance of routing.

We study the new and previous algorithms from the analytical perspective and through simulations. It shows that the average number of hops per query is improved by 15% to 25% in our algorithm. The load balancing scheme is based on multiple factors which could be optimized on cost, proximity, reputation and other factors. This scheme eliminates the need to periodically maintain metadata for load balancing. And it does not need a central pool available to maintain load information of overloaded peers and lightweight peers.

$$E = \sum_{p=1}^{P} \sum_{k=1}^{K} (\delta_{pk}^{o})^2 \qquad (1)$$

References

[1] M. Carey, S. Krishnamurthi and M. Livny, “Load Control for Locking: The ‘Halfand- half’ Approach”, in the 9th Symp. On Principles of Database Systems (PODS), April 1990.
[2] J. Chen, D. J. Dewitt, F. Tian and Y. Wang, “NiagaraCQ: A Scalable Continuous Query System for Internet Databases”. In ACM SIGMOD, 2000.
[3] P. K. Chrysanthis, “Transaction Processing in a Mobile Computing Environment”, in IEEE workshop on Advances in Parallel and Distributed Systems, October 1993.
[4] Dirckze, R. and L. Gruenwald, "A Pre-Serialization Transaction Management Technique for Mobile Multidatabases", ACM Mobile Networks and Applications, Volume 5, Number 4, December 2000, pp. 311-321.
[5] Dirckze, R. and L. Gruenwald, "A Toggle Transaction Management Technique for Mobile Multidatabases", ACM Conference on Information and Knowledge Management (CIKM), November 1998, pp. 371-377.
[6] M.H. Eich and A. Helal, “A Mobile Transction Model that Captures Both Data and Movement Behavior”, ACM-Baltzer Journal on Special Topics on Mobile Networks and App, 1997.
[7] B. Gedik and L. Liu. “PeerCQ: A Decentralized and Self-Configuring Peer-to-Peer Infrmation Monitoring System”. ICDCS 2003.
[8] G. Kortuem, J. Schneider, D. Preuitt, T. Thompson, S. Fickas and Z. Segall. “When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad-hoc Networks,” First International Conference on Peer-to-Peer Computing (P2P 2001), Linköpings, Sweden, August, 2001, pp. 75-91.
[9] S. K. Prasad, V. Madisetti, R. Sunderraman, et al. “System on Mobile Devices (SyD): Kernel Design and Implementation”, in First International Conference on Mobile Systems, Applications, and Services (MobiSys), Poster and Demo Presentation, May 5-8, 2003, San Francisco.
[10] S. K. Prasad, A. G. Bourgeois, E. Dogdu, et al. “Enforcing Interdependencies and Executing Transactions Atomically Over Autonomous Mobile

Data Stores Using SyD Link Technology", in Mobile Wireless Network Workshop held in conjunction with The 23rd International Conference on Distributed Computing Systems (ICDCS) , May 19-22, Providence, Rhode Island.

[11] S. K. Prasad, A. G. Bourgeois, E. Dogdu, et al. "Implementation of a Calendar Application Based on SyD Coordination Links", in the Third International Workshop on Internet Computing and E-Commerce in conjunction with the 17th Annual International Parallel & Distributed Processing Symposium (IPDPS), April 2003, Nice, France.

[12] A. Rao, K. Lakshminarayanan, R. K. Sonesh Surana, and I. Stoica. "Load balancing in structured p2p systems." In the 2nd International Workshop on Peer-to-Peer Systems (IPTPS), Feb. 2003.

[13] S. Ratnasamy, P. Francis, M. Handley, et al., "A Scalable Content-Addressable Network." In SIGCOMM 2001.

[14] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November 2001.

[15] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, Hari Balakrishnan. "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications." SIGCOMM 2001.

[16] System on Devices (SyD): A Model with Coordination Links and a Calendar Application, Utility Patent Filed,2002

[17] W. Tang, L. Liu and C. Pu, "Trigger Grouping: A Scalable Approach to large Scale Information Monitoring", In NCA 2003.

[18] W. Xie, S. B. Navathe and S. K. Prasad, "PeerDS: a scalable directory service", in submission.

[19] W. Xie, S. B. Navathe and S. K. Prasad, "Optimizing Peer Virtualization and Load Balancing", To appear in the 11th International Conference on Database Systems for Advanced Applications (DASFAA), Apr 2006.

[20] W. Xie, S.B. Navathe and S. K. Prasad, "Supporting distributed transactions over dynamically partitioned networks", in preparation.