Proximity Neighbor Selection in Structured P2P Network

Hancong Duan, Xianliang Lu
Dept. of Computer Science and Engineering,
University of Electronic Science and Technology of China
Chengdu, Sichuan, China 610054 86-28-83201924
{duanhancong, xlu}@uestc.edu.cn

Hui Tang, Xu Zhou and Zhijun Zhao
Institute of Acoustics, Chinese Academy of Science
Beijing, China 100080 86-10-82622482
{ tangh, zhoux, zhaozj}@dsp.ac.cn

Abstract

Structured P2P network offers efficient and fault-tolerant routing, object location and load balancing for upper applications. In this paper, we propose a novel scheme to improve the proximity property of structured P2P network. Basing on 2-dimension virtual network coordinates, the psychically nearby nodes are clustered into the same and adjacent regions after mapping from the network coordinates space to the identifier space of DHT. Through searching the corresponding region id using RPC calls in Chord, nodes obtain their nearby neighbors quickly in pure distributed way. Results obtained via simulation of large scale topology models denote that our scheme has lower average neighbor delay than randomly sampling approach and only incurs a modest additional searching overhead.

1. Introduction

The Widely-deployed applications of file sharing, overlay multicasting and web caching have motivated considerable studies into peer-to-peer system. Particularly in structured peer-to-peer overlay network, every node is assigned an identifier from the large identifier space by hashing nodes' IP, and the objects can be distributed to the nodes whose identifiers are equal or closest to the hash value of it. Several recent structured peer-to-peer overlay network systems, such as CAN[1], Chord[2], Pastry[3] and Tapestry[4], all can provide a scalable, fault-tolerant distributed hash table (DHT), where any object or node can be located within O(logN) routing hops by using small size routing table. The primary advantage of applying the hash function in DHT is load balancing and resilience. The objects can be distributed to the nodes on the overlay network uniformly. But on the other side, DHT also leads to the topology mismatches between the upper overlay network and the underlying IP network. As a result, a message may travel arbitrarily long distances in the Internet in each routing hop. It leads to the high stretch which increases the delay of publishing and locating an object.

To overcome the disadvantage above, it is the key problem to locate the nearby neighbors of nodes. In existing structured overlay networks, there have been three schemes： (1) the first is proximity neighbor selection (PNS), which is that the neighbors in the routing table are chosen based on their proximity; (2)the second is proximity route selection, which is the choice of next-hop when routing to a particular destination on the proximity of the neighbors; (3)The third is Proximity Identifier Selection (PIS)[5], where nodes can pick their identifier based on geographic location. Compared with other two schemes, PNS is more widely used method to decrease the routing stretch on structured peer-to-peer overlay network. So, this paper focuses on the PNS scheme.

We present a novel scheme for PNS. In our scheme, every node obtains a 2-dimension virtual network coordinates by using the distributed network coordinates system, Vivaldi[6]. Then, every node can find its nearby neighbors by employing mapping between the virtual 2-dimension network coordinates space and the identifier space in DHT. Furthermore,

Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)
0-7695-2687-X/06 $20.00 © 2006 IEEE

nodes being close to each other can be grouped into the same sector or adjacent sectors by calling RPC calls in Chord, such as Get()/Put(). Finally, entries of the node's routing table can be fast filled with physically nearby nodes, even there are the large scale nodes. To our best knowledge, the algorithm in this paper is the first one applying space partition scheme to optimize routing table in structured peer-to-peer overlay network.

The rest of this paper is organized as follows. Related work is discussed in section 2. In section 3, our PNS scheme is proposed. Section 4 presents the simulations. We conclude this paper in section 5.

2. Related works

In CAN[1], each node measures its network delay to a set of landmark servers to determine its relative position in the Internet and to construct an Internet topology aware overlay. This technique can achieve good performance, but it is not fully self-organizing. Moreover, it may cause significant imbalances which lead to the hotspots problem.

In Pastry[3], due to the churn of overlay network, a joined node should maintain the routing table, and displace the non-existing or distant node in the entry with the nearby one through exchanging the information of routing tables with the neighbors periodically. In Bamboo[7], this PNS approach is called neighbor's neighbor (NN) sampling. Just as analyzed in [7], under the high churn circumstance, the NN approach can't promise the properties of neighbor proximity in Pastry. Furthermore, Tapestry[4] applies the similar mechanism. To locate the nearby neighbor in the routing table entry, a node exchanges the routing information of its neighbor's inverse neighbor (NIN) for locating the nearby one.

Chord[2] is closely related to Pastry and Tapestry. In [8], The PNS algorithm based on global sampling is studied. However, the time for nodes to find their physically nearby nodes using global sampling is proportional to the size of the total overlay network. So, the performance of the PNS algorithm proposed in [8] is suboptimal.

As analyzed in [9], Chord applies ring geometry which has better flexibility to achieve proximity performance than ones applied in CAN, Pastry and Tapestry. So, our PNS scheme is based on Chord. Furthermore, since the experiment results in [7] indicate that the scheme of global sampling in [8] has the lowest neighbor delay among all PNS algorithms, we only compare the performance of our PNS algorithm with [8] in section 4.

3. Algorithm

Through Vivaldi, each node can obtain virtual 2-dimension network coordinates. The main steps of our PNS algorithm are as follows: first, we perform space mapping to make each node mapped to one region in identifier space of Chord without losing proximity relationship; the second is phase of searching nearby neighbors; finally, we use the obtained nearby neighbor set to optimize the routing table of Chord.

3.1. Space mapping

3.1.1. Partition virtual 2-dimension coordinates space. Once the node's virtual 2-dimension network coordinates are obtained, we divide the entire 2-dimension coordinate space C into many same area sectors by using concentric circle cluster denoted by O. O is constructed as follows: the radius and the origin of the first circle (denoted by c_1) is r and (0,0) respectively. The second circle c_2's radius is $2r$. The radius of c_3 is $3r$, and so on. As mentioned before, the entire 2-dimension coordinate space should be divided into same area sectors. We divide all annuluses between any pair of adjacent circles into sectors which have the same area as c_1 has. For convenience of describing, we number every annulus in the concentric circle cluster starting from c_1. The id of the area surrounded by most inner circle is r_1. The id of annulus outside r_1 is r_2, and so on. Sectors on r_i is denoted by $d_{i,j}$, $j \in \{1, 2, \cdots, n\}$,means $d_{i,j}$ is the jth sector in r_i.

Theorem1: $\forall d_{i,j} \in r_i, d_{k,l} \in r_k$, $r_k, r_i \in O$, if $S(d_{i,j})=S(d_{k,l})$, S is the area function, then $max(j)=2x-1$, $max(l)=2y-1$, $x, y \in \{1, 2, \cdots, n\}$.

Proof.

$$\because S(d_{1,1}) = \pi(xr)^2 - \pi[(x-1)r]^2 = \pi r^2(2x-1),$$

$$S(d_{i,j}) = S(d_{i,j-1}) = \cdots = S(d_{1,1}) = \pi r^2$$

$$S(r_i) = \max(j) \cdot S(d_{i,j}) = \max(j) S(d_{1,1}) = \max(j)\pi r^2 ,$$

$$\therefore \max(j) = 2x-1. \text{ In the same way, } \max(l) = 2y-1.$$

□

Theorem1 denotes that r_i should be partitioned to $2x$-1 sectors for guaranting the condition of each sector having the same area. Furthermore, assuming the nodes (or points) uniformly distribute in the 2-dimension coordinates space, sectors with same area make node clustering operation load balancing.

Fig 1 is an example of 2-dimension partitioned coordinates space. Every node locates at a sector based on its virtual 2-dimension network coordinates through

Vivaldi. Before mapping operation, we number every sector as follows: starting from inner sector to outer sector, we number the most inner sector (surrounded by c_1) $d_{1,1}$. For all other annuluses r_i, the sector id is numbered with $d_{i,1}, d_{i,2} \ldots, d_{i,2x-1}$ starting from the 0 degree along the counterclockwise direction. e.g. in r_3, there are five sectors which are numbered with $d_{3,1}, d_{3,2}, \ldots, d_{3,5}$ respectively.

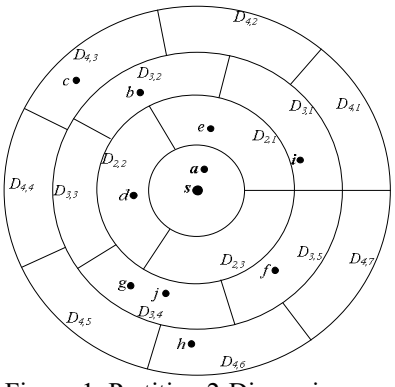

Figure 1. Partition 2-Dimension space

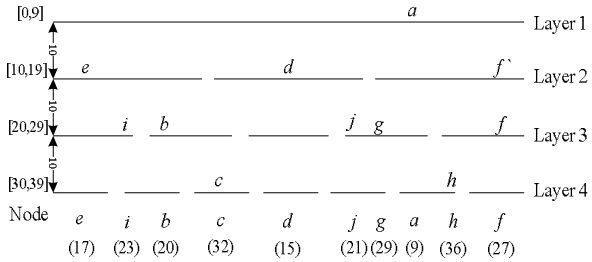

Figure 2. Mapping to multi-layer identifier space

3.1.2. Multi-layer node identifier space. Every sector in 2-dimension coordinates space is mapped to a region in the node identifier space in DHT uniquely. As every node can figure out its sector id in terms of network coordinates, it can find the region id in identifier space in pure distributed way basing on mapping rule. Furthermore, once nodes are in the same sector, they will be mapping to the same region in DHT. It is obvious that the proximity relationship between the nodes can be kept. Through Put(region.key) and Get(region.key), primary RPC call in Chord, nodes can locate its mapping region in Log(N) hops with high probability.

We define multi-layer node identifier space consists of independent identifier space with the same identifier capacity. The top or the first layer l_1 is corresponding to r_1 in 2-dimension coordinates space. The second layer (l_2) is corresponding to r_2, and so on. Furthermore, a logic notion named inter-layer distance is introduced, which is corresponding to the distance r between two adjacent concentric circles. Layer l_x consists of *2x-1* non-overlapping regions denoted by $I_{x,y}$, $x \in \{1, 2, \cdots, n\}$, $y \in \{1, 3, \cdots, 2x-1\}$. We number every region in multi-layer identifier space from top to down and from left to right. e.g. there are five regions in l_3, which are numbered with $I_{3,1}$, $I_{3,2}, \ldots, I_{3,5}$.So, region $I_{x,y}$ in l_x is corresponding to $d_{x,y}$ in r_i uniquely. Fig 2 depicts an example of 4-layer identifier space which is corresponding to Fig 1.

For instance, assuming origin point is *s*, and the radius is $r = 10$, *f* is the coordinates point of node F in 2-dimension coordinates space. After coordinates space is partitioned, *f* belongs to sector $d_{3,5}$ in terms of its distance (27) and angle ($7\pi / 4$) to origin point *s*. In the same way, as the inter-layer distance in multi-layer identifier space equals 10, *f* belongs to region $I_{3,5}$ using mapping ruler mentioned above. Once *f* obtains its region id, it can join that region by using Put(regionid.key)/Get(regionid.key) call in Chord. Because mapping keeps the proximity relationship, *f* can find its physically nearby neighbors in the same region or near regions in several routing hops. Since there are some drawbacks in previous PNS algorithm in DHT, the feature above encourages us to improve the PNS method in Chord.

3.2. Optimizing routing table

3.2.1. Searching for nearby neighbors. We use node *a* to explain the whole process of a node searching for its nearby neighbors. Firstly, node *a* can obtain its key_a, a unique ID in identifier space of DHT by using consistent hash function Hash(). In addition, it holds the region id after the space mapping operation mentioned above. Similarly, *a* use Hash(region.id) to get key_r , the region's key in identifier space. The active node called cluster node (CN) whose key is numerically closest to the key_r. CN is response for clustering the nodes which are in the same sector or region of *a*. Node *a* can get CN's address by using Get(key_r) RPC call in Chord. Through calling Put(key_r, $info_a$), node *a* publishes its information (including virtual 2-dimension network coordinates, identifier, IP, Port) to CN for registering itself to region $I_{x,y}$. Since nodes belonging to the same region $I_{x,y}$ register their information on CN, *a* can obtain its previously joined nearby neighbors from CN. Once *a* finds neighbors in region $I_{x,y}$, it adds them into its neighbor set for future routing table optimizing.

But if there isn't any node having joined region $I_{x,y}$ in layer l_x when *a* joining, *a* begins expanding searching as follows: *a* searches nearby neighbors in other regions which are around $I_{x,y}$, such as $I_{x,y-1}$, $I_{x,y+1}$, $I_{x-1,j}$, $I_{x-1,j+1}$, $I_{x+1,k}$, $I_{x+1,k+1}$. In the same way, *a* locates *CN* in each neighbor regions through calling Get(key_*) six

times at most, and key_* is one of the keys of neighbor regions IDs. If any neighbor node in six regions is found, the searching process is finished. Otherwise, node *a* continues searching in the upper layer identifier space until finding its neighbor node or reaching the first layer. It should be cared that one region $I_{x,y}$ may cover two regions in layer l_{x-1} simultaneously. So, the two regions are all upper regions of $I_{x,y}$. Along the searching path from the original region $I_{x,y}$, node *a* registers itself to every *CN* in the regions. Once new node joins one of regions on the path, node *a* is notified. Then node *a* will determine whether add the new node into its nearby neighbor set.

3.2.2. Optimizing Entries in Chord Routing Table. In Chord routing table, the entry of the *i*th finger needs be filled with one live node whose identifier immediately follows $k+2^{i-1}$, *k* is the node identifier. [9] applied more flexible routing table constructing strategy that any live node with lower delay whose identifier is in $[k+2^{i-1}, k+2^i)$ can be filled in the entry of *i*th finger. Although the random sampling neighbor approach in [8] improves the routing performance, just as analyzed in [7], its results are suboptimal at the expense of periodically heavy probing. In our scheme, node *a* keeps all found nearby neighbor information in its neighbor set during the previous lightweight searching. So, we can use it to optimize the routing table directly.

Suppose node *a* has neighbor set N_a, $n_i \in N_a$, n_i is a nearby neighbor in N_a. *a* calculates which finger entry should n_i be in. After that, node *a* compares the delay to n_i and current node which is in the finger entry of n_i . The node with lower delay is picked, and filled in this finger entry. As the improved version Chord, information of several backup nodes is stored in each entry for decreasing the impact of nodes churn. The non-picked node is added into the backup node list of this entry. After all nodes in N_a are checked, the rest entries to which none of n_i belong are optimized as follows. Node *a* will ask all nodes in N_a for their routing tables and extract the successor from each finger entry. Cause every node contains its neighbor network coordinates, so *a* can get the distance (delay) to each successor just by calculating Euclidean distance without physically probing. Similarly, *a* chooses the low delay node as its new finger successor in the same way described above.

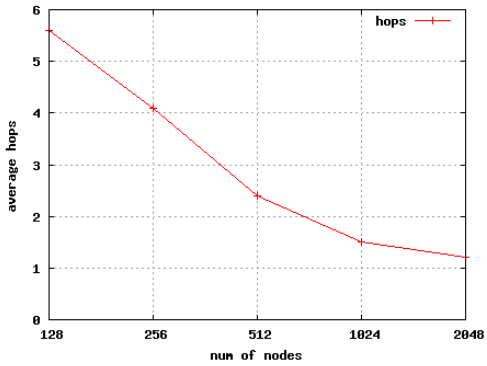

Figure 3. Average hops of joining region

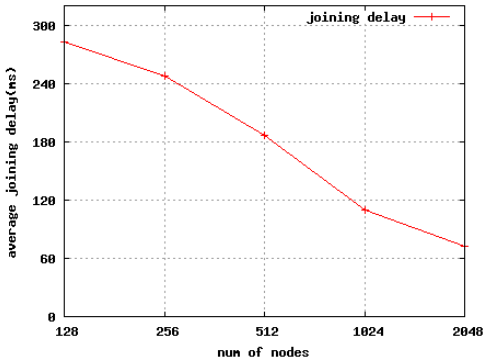

Figure 4. Average delay of joining region

4. Simulations and discussions

We use p2psim[10], an event-driven p2p simulator designed by MIT Pdos Lab, to investigate and compare the performance of our scheme and [8] under different scenarios. In simulations, the network topology applies the E2E graph defined in p2psim. The network graph size is 128, 256, 512, 1024 and 2048 respectively. Nodes join one by one every 10ms interval. The delay matrix applies King data set[11] which is based on the Internet measurements by analyzing the multiple DNS server latency. The average delay in all graphs is 154ms. This simulation condition is as same as the one in [8] for keeping justice. In the following simulation, the inter-layer distance value is 60ms. For convenience of depicting, the proximity neighbor selection approaches in [8] and in this paper are denoted by PNS(16)(randomly sampling 16 nodes every turn) and DHT-PNS respectively.

Firstly, we evaluate the payload of the node joining its corresponding region in multi-layer identifier space after space mapping. Since we use Chord's routing

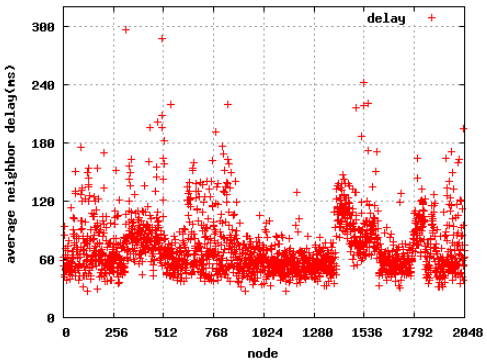

Figure 5. Distribution of nodes average delay in PNS(16)

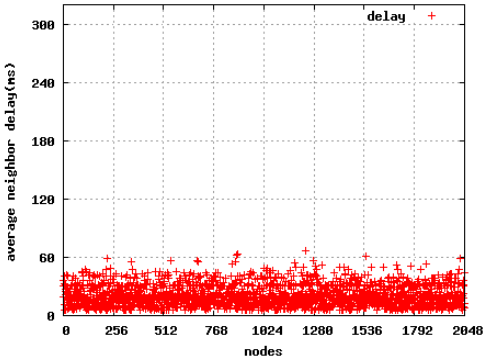

Figure 6. Distribution of nodes average delay in DHT-PNS

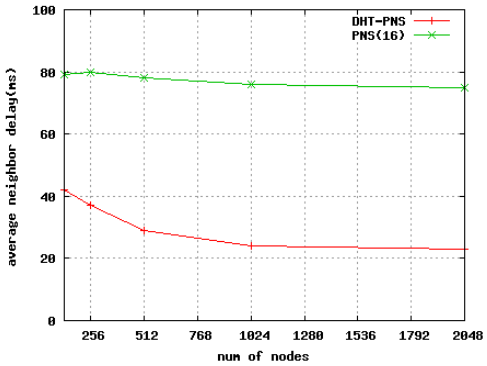

Figure 7. Average neighbor delay

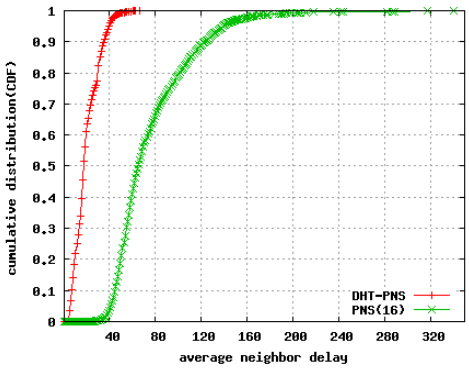

Figure 8. CDF of average neighbor delay(2048nodes)

algorithm to search nearby neighbor along the path to the origin point, it's important to know the payload and the consumed time for a searching. We record the hops number and the searching time. In Fig 3 and Fig 4, when there are 128 nodes, average hops number and delay are 5.6 and 283ms respectively. When network size is enlarged, the joining hops number and delay reduce to 1.2 and 72ms when there are 2048 nodes in Chord. This is because the node can find nearby neighbors in the origin region with high probability when network size increased. Thereby, the results indicate that our search approach only puts low payload to Chord and has high scalability when network size is large.

Now, we begin to investigate the performance of our proximity neighbor selection algorithm, DHT-PNS, by comparing with PNS(16) proposed in [8]. Fig 5 and Fig 6 depict the distribution of neighbor delay of each node when there are 2048 nodes in PNS (16) and DHT-PNS. The average neighbor delay are 75ms in PNS(16) and 23ms in DHT-PNS respectively. Furthermore, the results show that DHT-PNS has better convergence property of delay distribution than PNS(16). As analyzed in previous section, DHT-PNS make the nodes find each other in the regions on the path to the origin point with high probability. This searching approach also provides a clustering mechanism for nodes to fill their routing table entries with nearby neighbors quickly. From Fig 5, a suboptimal result is obtained through randomly sampling 16 nodes in PNS(16).

Fig 7 compares the average neighbor delay between DHT-PNS and PNS(16) when there are 128, 256, 512, 1024, 2048 nodes in Chord. For PNS(16), the average neighbor delay decrease slightly when enlarging network size. The reason is that PNS(16) selects the low delay ones as neighbors after randomly sampling 16 nodes. But its effect is limited when network size is large. However, since the proximity relationship between nodes is protected, DHT-PNS can apply Put()/Get() RPC calls to cluster nearby neighbors efficiently. The average neighbor delay decrease from 43ms (128 nodes) to 23ms(2048 nodes). Fig 8 gives the cumulative distribution function comparison of average neighbor delay when applying DHT-PNS and PNS(16) for optimizing Chord routing table. The average neighbor delay of 90 percent of nodes is less than 40ms in DHT-PNS. But for PNS(16), it increase to 120ms. This result proves DHT-PNS has better optimizing performance than PNS(16).

5. Conclusions

In this paper, we proposed a novel proximity neighbor selection scheme (DHT-PNS) for structured P2P network based on the network coordinates and space mapping. In our scheme, we first partition the entire 2-dimenstion coordinates space into multiple sectors with the same area by a cluster of concentric circles. Nodes determine itself sector id in terms of its virtual 2-dimension network coordinates and get the corresponding region id with mapping rule in pure distributed way. By calling several operations of Get(), nodes can obtain physically nearby neighbors through the cluster node (CN) which is responsible for region id. So, the entries of routing table of nodes can be optimized. From the simulation results, when there are large scale nodes, our scheme, DHT-PNS, has lower average neighbor delay than one in [8] with reasonable searching payload instead of heavy probing.

References

[1] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of* SIGCOMM'01, San Diego, CA, 2001.

[2] 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 the ACM SIGCOMM '01*, San Diego, California., 2001.

[3] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of Distributed Systems Platforms (Middeware)*, Nov.2001.

[4] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," *UC Berkeley UCB/CSD-01-1141*, April 2001.

[5] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," New York, NY, United States, 2002.

[6] D. Frank, C. Russ, K. Frans, and M. Robert, "Vivaldi: a decentralized network coordinate system," in *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*. Portland, Oregon, USA: ACM Press, 2004.

[7]D. G. Sean Rhea, Timothy Roscoe, and John Kubiatowicz, "Handling churn in a DHT," *Proceeding of the USENIX Annual Technical Conference*, June, 2004.

[8] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *Proceedings of USENIX Symposium on Networked Systems Design and Implementation*, June, 2004.

[9] R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of dht routing geometry on resilience and proximity," in *Proceedings of ACM SIGCOMM*, 2003.

[10] p2psim.[online] http://pdos.csail.mit.edu/p2psim/.

[11] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating Latency between Arbitrary Internet End Hosts," in *Proceedings of ACM SIGCOMM,* 2002.