Analysis and improvement on Chord protocol for structured P2P

Shunli Ding
Northeastern University at Qinhuangdao
Qinhuangdao, China
dingsl@163.com

Xiuhong Zhao
Qinhuangdao Entry-Exit Inspection and Quarantine Bureau
Qinhuangdao, China

***Abstract*—The key problem in P2P network is effectively to locate resources. Chord has been widely used as a routing protocol in structured P2P networks, but there is a lot of redundant information in the node's finger table. By analyzing Chord protocol, an improved Chord algorithm is proposed in this paper. Analysis on theory and simulation results show that the improved finger tables reduce redundancy and improved Chord algorithm has better performance.**

Keywords-structured P2P network ; Chord protocol; routing protocol; finger table

I. INTRODUCTION

In recent years, peer-to-peer (hereafter P2P) systems have been the burgeoning research topic in large distributed system. This popularity can be seen as a result of the features of these systems such as scalability, node autonomy, self-configuration and decentralized control[1]. P2P systems are classified into unstructured P2P systems, such as Gnutella[2], and structured P2P systems or Distributed Hash Tables (DHTs), such as Chord[3]. Structured P2P network is becoming popular for their advantages of high scalability and good performance. The main advantage of DHTs is that data placement and search procedures generate less traffic. Among the DHT-based approaches, Chord is one of the most popular systems. As structured P2P networks most representative network architecture protocol, the Chord protocol has not only simple design idea and also good distribution, scalability, stability and load balancing. There are a lot of research and improvement on Chord to meet the actual demand[4-6].

In this paper, we mainly focus on modification of Chord in order to improve its performance. We adopt removing redundant information in finger tables to improve query efficiency and successfully implement this mechanism in p2psim simulator. The simulation showed significant performance improvement of the improved Chord protocol.

II. ANALYSIS OF CHORD PROTOCOL

A. Architecture

Chord is a resource location and lookup algorithm based on distributed hash function (DHT). A distributed hash table stores key-value pairs by assigning keys to different computers (known as "nodes"); a node will store the values for all the keys it is responsible for. Keys and nodes on behalf of the resources information are mapped to a virtual space. Each network node is assigned a unique node identifier (*nodeID*) using a hashing function, that is, *nodeID=H(the attributes of node)*, where *H* is hash function and the attributes of node include IP address, port number, public key, and random number or their combination. The keys and the node identifiers are arranged on an identifier circle of size 2^m called the Chord ring, where *m* is the number of bits in the identifiers. The identifiers on the Chord ring are numbered from *0* to 2^m-1. Keys are assigned to the first node in the identifier space that has in identifier equal to or higher than its own (modulus 2^m). This is known as the successor node. Every node stores details of its own successor.

The Chord protocol supports just one operation: given a key, it will determine the node responsible for storing the key's value. Queries are passed around the circle via successor pointers and require traversing all nodes to find the appropriate mapping[7].

Each node maintains a routing table with size log^2N, called the finger table, where *N* is the total number of the nodes. Chord is to improve the stability of the system and seeking efficiency with redundancy. When the system was in stable condition, each node maintains only *O(logN)* other node's information, and lookup just needs to send *O(logN)* messages to other nodes. When a node joins and leaves, the system volume of messages generated is not over $O(log^2N)$ in most cases.

B. Key location algorithm

The core function of the Chord protocol is the key location function. When a node is asked to find a certain key, it will determine the highest predecessor of this key in its routing table and forward the key to that node. The lookup time is *O(log N)*, since the query is forwarded at least half the remaining distance around the ring in each step.

The key location algorithm includes three functions: *n.find_successor(id)* (ask node n to find id's successor), *n.find_predecessor(id)* (ask node n to find id's predecessor) and *n.closest_preceding_finger(id)* (search the local table for the id's closest predecessor).

C. Node Joining algorithm

In dynamic network, a node can at any time join to or leave from network. The main challenge that performs these operations is to maintain the position of each node in the network. To achieve this goal, Chord protocol must ensure that each node's successor is always correct. The function *successor(k)* is responsible for the index of node *k*. When a node joins the system, the successor pointers of some nodes will have to change. It is important that the successor

978-1-61284-486-2/11/$26.00 ©2011 IEEE

pointers are up to date at any. When node *n* want to join the network, node *n* will call function *n.join(n'),* where *n'* is a node gained by some way in Chord ring and node *n* will ask *n'* to find its successor.

D. Adaptive algorithm of node

To solve the failure of the routing tables caused by concurrency and abnormal operation, Chord protocols takes adaptive algorithm to ensure the correctness of the successor. The adaptive algorithm includes four functions: *join(n')*, *stabilize( )*, *notify (n')* and *fix_fingers( )*. Chord system finds the successor of node *n* by an existing nodes *n'*. Each node periodically calls the stability function *stabilize( )* to amend its successor. The *notify (n')* to notice its successor to amend precursor node. The *fix_fingers( )* random revises its routing table.

E. Replication and cache

Chord ring keeps the correctness of the successor node by redundancy. Each node maintains a successor list (finger table) to save *r* successors of the node in Chord ring. Unless *r* successors are all failure, the relation between the node and its successors can be amended. Typically, set *r=O(logN)*. Assuming that each node failure probability is *1/2*, function *find_successor( )* can still find correct successor in *O(log N)* hops in this case dynamic.

Replication mechanism will copy the data objects saved by node *n* to *r* successors of node *n*, which enhances data availability and persistence. Cache mechanism of Chord will buffer paths. Each intermediate node buffers gained data objects in location process to improve access to data speeds.

F. Existing Problems

Three mechanisms, the routing table, replication and cache, improved greatly the fault tolerance, data availability and locating efficiency, but there also are some problems in Chord. The consistent hash function does not take into account the physical network characteristics, for example, some nodes with very close identifier may be very far apart on the physical network, thus lead to the logical network no match the physical network. In the process of data query, Chord algorithm takes hops as the cost of the query and don't consider factors such as network latency, so some paths with logically optimum cost may really not be optimal. In the process of Chord network stability, although the system stability being maintained, too many messages will occupy the network bandwidth and influence on network utilization.

Aiming at existing redundant information in the Chord finger table and without considering network delay factors to cause large delay nodes, the paper presents improvement measures to enhance the efficiency of Chord network.

III. THE IMPROVEMENT OF CHORD ROUTING ALGORITHM

A. Routing Information

Each node *n* maintains a finger table with up to *m* entries. The *ith* entry in the table at node *n* contains the identifier of the first node *s* that succeeds *n* by at least 2^{i-1} on the identifier circle ($s= successor(n+2^{i-1})$). *s* is called the *ith* finger of node *n*.

There is large number of redundant information in the Chord finger table and each lookup may span half of the address space. If want to query a keyword in the other half of the Chord ring, first routing algorithm will query on the half Chord ring of the source node, and then go to query the other half ring, obviously, which will drop the query efficiency. Another problem, if a source node want query the destination node that is the precursor of its predecessor or is on the address space between last node and its predecessor, the query message must jump a circle to find destination node, which greatly waste of the query time.

1) Analysis on redundancy of node in the Chord finger table

Assuming a Chord ring size is 2^m, and the total number of the node is *N*. For a node *V* in the ring, its finger table size is *m*, and denoted as $V_1, V_2, ..., V_m$. According to buildup rules of the routing table, the region of selecting successor corresponding to *ith* item is $[(V+2^{i-1})mod(2^m-1), (V+2^i)mod(2^m-1))$, that is, the region size is 2^{i-1}, denoted by *d(i)*. Therefore, $d(1)=2^0, d(2)=2^1,..., d(m)=2^{m-1}$. It can be seen that this is a geometric sequence, so the first several data is very small and it is easy to have the same successor, that is, lead to redundant. Chord ring distributes individual nodes based on DHT, so the distance between nodes is approximately equal. The average distance of the entire loop is $d_{avg}=2^m/N$. Assumes that there is an integer *k* as so to 2^k approximately equal to *N*, so $d_{avg}=2^m/2k=2^m-k$. Obviously *k* cannot be greater than *m*. Moreover, according to DHT principle, it is easy to conflicts if *m=k*, so *m* should be far greater than *k*.

Definition 1: Distance. The number of identifiers (or nodes) between two identifiers (or two nodes) is called distance, denoted as *d*.

Definition 2: Redundancy. Redundancy means the number of same successor (or predecessor) in the Chord finger table, denoted as *R*.

In ideal circumstances, that is, even the ring is full it will not produce the conflicts, we can approximate think that when $d_{avg}=2^i$ (i.e. *m-k=i*) node *V* will produce *i-1* redundant information. That's because the average distance in Chord ring is 2^i and the front *i-1* item is also a geometric progression. According to geometric progression summation formula, knows $S(i-1)=2^i-1$.So $S(i-1)<d_{avg}$, that means, when the front *i-1* items search for successor, the total lookup interval is less than d_{avg}. Therefore, the successors of front *i-1* items all are in *ith* interval, but the successor of *ith* item is uncertainly in this region.

2) Modify the redundant information in the finger table

Through the above analysis we can clearly see that there is a large of redundancy in Chord finger table. To enhance the efficiency, it is necessary to modify the redundant information in the finger table. Assuming the front *i* items in routing tables have the same successor, the front *i-1* items need to be modified. The steps are as follows.

Step 1: the successor lookup region of *ith* item is $[(V+2^{i-1}) mod (2^m-1),(V+2^i) mod (2^m-1))$. Merges it with the front *i-1* items to region $[V+1, (V+2^i) mod (2^m-1))$, but successor unchanged.

Step 2: redistribute the space of the front *i-1* items. The search for successor changes to search for precursors. Looking for precursor space of first item is $[(V-2^0+2^m) \bmod (2^m-1), (V-2^1+2^m) \bmod (2^m-1)), \ldots$, and looking for precursor space of *ith* item is $[(V-2^{i-1}+2^m) \bmod (2^m-1))$.

Step 3: lookup predecessor in new space updated by step 2 and compare them. If there aren't the same predecessor updating end, or continue to merge the region with the same predecessor from step1.

If the front *i-1* items in routing tables have the same successor but the successor of *ith* item is different from them, the front *i-2* items need to be modified. The method is the same as above three steps.

3) Predecessor to achieve the farthest distance

Known $R=i-1$, the space size of the *i-1* entries is, in order, 2^1-2^0, 2^2-2^1,..., $2^{i-1}-2^{i-2}$. The total amount of the front *i-1* entries space is, the sum of the geometric sequence, $2^{i-1}-2^0=2^{i-1}-1$. Through a merger of third step, the space size of *i-1* entries is, in order, 2^i-2^{i-1}, $2^{i+1}-2^i$, ..., $2^{2*i-3}-2^{2*i-4}$. Therefore, the space of the last node looking for predecessor is $[(V-2^{2*i-4}+2^m) \bmod (2^m-1), (V-2^{2*i-3}+2^m) \bmod (2^m-1))$ and achieving the farthest bits in the identifiers is $(V-2^{2*i-3}+2^m) \bmod (2^m-1)$. For node *V*, on clockwise and counter-clockwise direction, achieving the farthest distance is respectively $2^{2i-3}$and 2^{m-1}.

Conclusion 1: On the Chord ring with redundancy $R=i-1$, achieving the farthest node is $V+2^{m-1}$ in clockwise direction and $(V-2^{2*i-3}+2^m) \bmod (2^m-1)$ in counter-clockwise direction.

Conclusion 2: Achieving the farthest distance is $ds=2^{m-1}$ in clockwise direction and $dn=2^m-2^{2i-3}$ in counter-clockwise direction.

Conclusion 3: For the same node, the ratio of achieving the farthest distance in counter-clockwise direction and in clockwise direction is $t=dn/ds=(2^m-2^{2i-3})/2^{m-1}=2-2^{2i-m-2}$.

4) The rule of routing direction selecting

For improved finger table, a node routing direction is changed from clockwise to both clockwise and counterclockwise. Therefore when routing in the light of a given finger table, it first is necessary to judge direction. For example, for the finger table of node 0 (as described in the following table 1), lookup ends at key 110 in clockwise direction and at key 191 in counterclockwise direction.

TABLE I. THE MODIFIED FINGER TABLE OF NODE 0

No.	Interval Size	Interval	Successor
1	15	[239, 254)	240
2	16	[223, 239)	225
3	32	[175, 207)	191
4	15	[1, 16)	8
5	16	[16, 32)	32
6	32	[32, 64)	50
7	64	[64, 128)	110
8	128	[128, 0)	240

For a given target node, there are three states. First, clockwise find directly if the identifier is equal or lesser than 110. Second, counterclockwise find directly if the identifier is equal or greater than 191. Third, the identifier is more than 110 and lesser than 191. According to the above conclusion 3, set the ratio of achieving the farthest distance in counter-clockwise direction and in clockwise direction is *t* and identifier or node $S=(191-110)\times t+191=81t+191$, so counterclockwise sends routing if given target node is less *S*, or clockwise sends.

B. The finger table with network latency

Pure Chord algorithm is running on the network application layer, so the topological structure of nodes is transparent to the algorithm. In a real application, this operating mode can't make lookup to achieve optimal, because the lookup of adjacent nodes on topological structure may be forward to other nodes that are very far on physical, consequently the network latency in routing process greatly increased.

In order to decrease network latency, a column is added in the finger table of node to record the network latency from the node to other nodes. When a node joins the Chord ring, the message that the join algorithm sent need to add a timer so as to record the latency (*NL*) and save it in the table. After the finger table has updated, calculate the average of these NL_s, denote as NL_{avg}. In the process of sending routing in the future, it may consider sending the routing to node's previous item or later item if the *NL* of a node is far great than NL_{avg}.

The searching procedure of improved algorithm:

Step 1: When a node lookup the next hop routing nodes (assuming, is *ith* item) in its finger table, lookup end if there is the target node, or turn to step 2.

Step 2: Comparing the network latency *NL(i)* from the node to next hop node with NL_{avg}, if $NL(i) <= NL_{avg}$, send querying message to the corresponding node and turn to step 1, or turn to step 3.

Step 3: if $NL(i) >> NL_{avg}$, send the querying message to nodes correspond to the *(i-1)th*, *ith* and *(i+1)th* entry, and turn to step 1.

C. The maintenance algorithm of improved Chord protocol

1) Object location algorithm

The object location algorithm of improved Chord protocol is illustrated in the following figure 1.

First, when a node ask node *n* to query the successor of *id*, compare *id* with the farthest predecessor and successor in the finger table of node *n*. According to the rule of routing direction selecting, select the optimal direction to route. Second, when looking for the next closest node in the routing table of node *n*, at the same time comparing the *NL* of node, if the *NL* is far greater than the average *NL*, then forward to the node with smaller *NL* among the previous and the next node.

2) The node joinning algorithm

The node joinning algorithm in improved Chord protocol mainly includes functions *n.join( )*, *n.init_finger_table( )*, *n.update_others( )*, *n.update_finger_table( )* and *n.f_finger_table( )*. Where, function *n.join( )* calls function *n.init_finger_table( )* to initialize node finger table and calls function *n.update_others( )* to update other nodes information, and final copies nodes that their successor

node's *id* locates in interval *(predecessor,n]* to the finger of node *n*. The function *n.update_finger_table(s,i)* is responsible for updating the finger table of node *n* with *s* if *s* is the *ith* item. The function *n.f_finger_table( )* is responsible for modifying redundant information in the light of method in 3.1 section, as shown in the following figure 2.

```
// Ask node n to query the successor of id
n.find_successor(id)
{ n'=find_predecessor(id);
  return n'.successor
}
// Ask node n to query the predecessor of id
n.find_predecessor(id)
{ n'=n;
   if (|P-id|>|S-id|)
   //P and S is respectively the farthest predecessor and
successor of node n
  {  while (id∉ (n' , n'.successor])
          n' = n'.closest_preceding_finger(id);
    return n'; }
  else
    {  while (id∉[n'.predecessor , n/))
       n'= n'.closest_preceding_finger(id);
    return n';    }
}
// Return the closest finger preceding id
n.closest_prededing_finger (id)
{ for i=m to 1
   if(finger[i].node∈(n,id) and NL(i)<k*NL_avg)
     return finger[i].node;
  else if (NL(i-1)< NL(i+1))
      return finger[i-1].node;
    else  return finger[i+1].node;
   return n;
}
```

Figure 1. the main function of key location algorithm

IV. THE ANALYSIS OF SIMULATION EXPERIMENT

To show the correctness and effectiveness of the algorithm, we performed simulation on improved Chord protocol using the network simulation tool P2PSim and compared with Chord.

Simulation experiment used redhat9.0 operating system. The size of Chord ring is 2^{11} and the maximum capacity is 2048 nodes. The experiment takes 9 Chord networks to get average routing hops of each network. The number of nodes of each network is respectively 100, 200, ..., 900. In simulation for each network, each node queries a random key for 100 times. Finally calculate the average lookup hops of all nodes.

We evaluated and compared the algorithm using the following metrics: average routing hops, the average routing delay and the number of bring about a node joining in ring.

A. The average routing hops

The routing hops of searching node is an important parameter of measuring quality of service of Chord protocol, which can reflect the efficiency of a P2P network based on Chord. Figure 3 shows comparison for the average routing hops of improved Chord and the original Chord protocol under the different node sizes.

```
// modifying redundant information
n.f_finger_table( )
{  R=0;
   for i=1 to m-1
   if(finger[i].node=finger[i+1].node)
        R++;  // compute the redundancy of the routing table
     finger[i].interval=[finger[1].start,finger[i+1].start);
     finger[1].node=n.predecessor;
    t=1;
     for i=1 to R-1
     { j=i+1;
      finger[j].node=n.find_predecessor(finger[t].strat);
      while(k<R-2)
         if(finger[j].node==finger[i].node)
         finger[j].node=n.find_predecessor(finger[++t].strat);
}
}
```

Figure 2. the main function of joinning algorithm

It can be seen that the average routing hops of the original Chord is about *(log N)/2* that is exponential growth. The average routing hops of improved Chord has been greatly reduced than the original Chord because introduced some precursor nodes as removing redundant information in the routing table to avoid around the entire ring to search. However, with the increase in the number of nodes, the effect of improving gradually diminished because the augmentation of the number of nodes caused the reduction of redundancy information in the finger table and then make routing to the biggest predecessor node been bigger, so lead to widen the blind spots of the entire finger table. However, the decline did not wear away the role of the improved Chord. It is because the ratio of the bits of identifier and the number of nodes is huge in order to avoid conflict of hash property values of nodes.

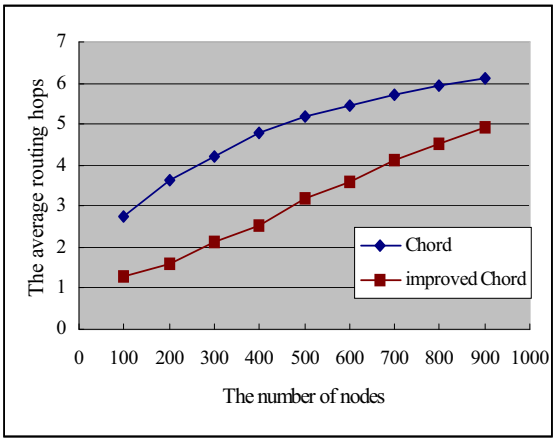

Figure 3. The average routing hops of two algorithms

B. *The average routing delay*

Reducing the average routing hops of querying process is only a means, the ultimate aim is to reduce the average routing delay. Figure 4 illustrates comparison for the average routing delay of improved Chord and the original Chord protocol under the different node sizes.

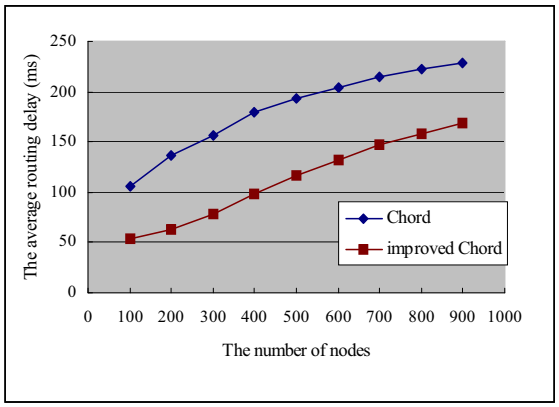

Figure 4. The average routing delay of two algorithms

From the diagram it can be noted that for improved Chord algorithm much improvement has been made in routing delay and improvement effect is more obvious with the increase of the number of nodes. Reduction of the average routing hops would lead to reduction in the average routing delay. Furthermore, the routing information is forwarded along the nodes of the minimum delay.

V. CONCLUSION

In this paper we proposed an improved method on Chord protocol. We performed a simulation study and compared with improved Chord and the original Chord protocol. Simulation results show that improved Chord can reduce the average routing hops and the average routing delay. The experiment demonstrated the correctness and effectiveness of the improved Chord algorithm, so the algorithm can improve the performance of P2P networks and be better suited for P2P network.

REFERENCES

[1] Raddad AI King, Abdelkader Hameurlain, and Frank Morvan, "Query Routing and Processing in Peer-to-Peer Data Sharing Systems," International Journal of Database Management Systems (IJDMS), Vol. 2, No. 2, May 2010.

[2] Gnutella http://www.gnutella.com

[3] Stoica Ion et al. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," Proceedings of SIGCOMM'01 (ACM Press New York, NY, USA), pp. 149–160, 2001.

[4] Zolt´an Lajos Kis , R´obert Szab´o, Chord-Zip: A Chord-Ring Merger Algorithm[J] , IEEE COMMUNICATIONS LETTERS , 2008, 12(8) : 605–607

[5] Giang Ngo Hoang, Hung Nguyen Chan, et al. "Performance improvement of Chord Distributed Hash Table under high churn rate," International Conference on Advanced Technologies for Communications, pp:191–196, 2009.

[6] R. Cuevas, M. Uruena, A. Banchs, "Routing Fairness in Chord: Analysis and Enhancement," 28th IEEE Conference on Computer Computer Communications - INFOCOM 2009, IEEE, pp:1449–1457, 2009.

[7] http://www.inf.ed.ac.uk/teaching/courses/ip/chord-desc.html