CNP: A Protocol for Reducing Maintenance Cost of Structured P2P

Yu Zhang, Yuanda Cao, and Baodong Cheng

Beijing Laboratory of Intelligent Information Technology,
School of Computer Science, Beijing Institute of Technology, Beijing 100081 PRC
zhangyubit@gmail.com, ydcao@bit.edu.cn, cbd@bit.edu.cn

Abstract. With highly dynamic, structured P2P system needs very high maintenance cost. In this paper we propose a Clone Node Protocol to reduce the maintenance cost of structured P2P system by a mechanism of clone nodes. In order to verify the efficiency of CNP, we achieve a Clone Node Chord structure based on CNP, i.e. CNChord. Furthermore, we implement a bidirectional CNChord (BCNChord) in order to reduce query time of CNChord. Theoretical analysis and experimental results show that CNChord can greatly reduce the cost of maintaining P2P structure and BCNChord can effectively improve the query speed. In a word, CNP can effectively reduce the maintenance cost of structured P2P.

Keywords: maintenance cost, structured P2P, CNChord, BCNChord.

1 Introduction

Although structured P2P systems have many advantages, like scale-free topology, resilience, self-organization and equalized distribution of ID space, etc. Nodes in the system need to use more bandwidth and high cost to maintain the structure.

There is few research on maintenance cost of structured P2P system itself, most of researches are about improving query efficiency by change the structure of P2P overlay network, such as various transforms of Chord[1]. Karger [2] proposed a version of the Chord peer-to-peer protocol that allows any subset of $Node_s$ in the network to jointly offer a service without forming their own Chord ring. Montresor[3] introduced T-Chord, that can build a Chord network efficiently starting from a random unstructured overlay. After jump-starting, the structured overlay can be handed over to the Chord protocol for further maintenance. Zols [4] presented the hybrid chord protocol (HCP) to grouping shared objects in interest groups and use it In mobile scenarios. Joung[5] proposed a Chord2 structure,they found $Node_s$ in P2P systems are not equivalent, Some $Node_s$, known as "super peers", are more powerful and stable than the others. They use super peers to serve as index servers for query, and to speed up query. Kaiping[6] designed a FS-Chord structure. EpiChord[7] is a DHT lookup algorithm that demonstrates that we can remove the O(logn)-state-per-node restriction on existing DHT topologies to achieve significantly better lookup performance. Approaches in [8-10] adopted physical topologies to reduce maintenance cost.

In fact the maintenance of structure is very large, especially in highly dynamic P2P network. The maintenance may affect the usability of system. This article focuses on reducing maintenance cost of structured P2P system itself, and proposes a new Clone Node Protocol, CNP for short.

Chord is one of the most quoted structured P2P systems. In this paper we take Chord for example, realized a Clone Node Chord based on CNP, CNChord for short. Moreover, the route table of CNP is improved to support bidirection searching and CNChord with this improvement is called BCNChord.

2 Clone Node Protocol

2.1 Protocol Description

The basic idea of CNP is to clone all nodes in structured P2P system. The clone node will take part of the original node after the original node leaves the system. When the original node joins the system again, the changes will be synchronized from clone node to the original node.

Firstly we take Chord for example, realized a Clone Node Chord based on CNP, short for CNChord.

2.2 Structure of CNChord

Definition 1. *Main Node: Every node in the system is assigned a basic role, which is called Main Node, noted by Node,e.g.,*$Node_x$.

Definition 2. *Successor:* $Node_x$*'s successor node, say* $Node_s$*, is the first node along the clockwise direction in the Chord ring which is behind* $Node_x$*, expressed as successor(*$Node_x$*)=*$Node_s$*. As a network with n* $Node_s$*, firstly, every node is composed to Chord ring. In the sequence of clockwise, node is denoted by* $Node_1, Node_2, \cdots, Node_n$*. For any* $Node_x$*,* $1 \leq x \leq n$ *,if successor(*$Node_x$*)=* $Node_s$*, then* $Node_s$ *is cloned to* $Node_x$*, so that* $Node_x$ *will keep both the information of* $Node_x$ *and* $Node_s$*. After the process above, CNChord is established.*

Definition 3. *Predecessor: An predecessor node of* $Node_x$*, called* $Node_p$*,which is the first node along the anti-clockwise direction in the chord ring, presented as predecessor (*$Node_x$*)=*$Node_p$*.*

Definition 4. *Clone Node: In the process above, after Nodes is cloned to* $Node_x$*, the mirror node of Nodes is called Clone Node, denoted by CNode, such as CNodes means Clone Node of Nodes.*

Figure 1 shows a typical Chord structure with 4 nodes, and the corresponding structure of CNChord. Main nodes are drawn in black, 'N' is the abbreviation of the word "Node". While clone nodes are drawn in gray, 'C' is the abbreviation of the word clone.

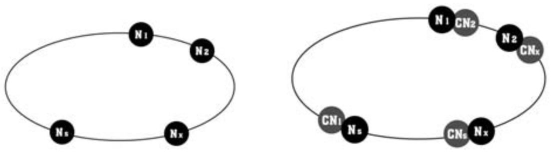


Fig. 1. (a) Chord with 4 nodes. (b)CNChord corresponding to (a).

2.3 Initialization Process

CNChord adopt similar structure with Chord's ring. When setting up CNChord, nodes compose a Chord ring according to[1]. Then every node performs clone node progresssee algorithm 1 to establish CNChord overlay network. If lots of nodes in Chord perform clone node process simultaneously, it will raise a large amount of network information of replication, this may make message flow increase in a short period, even block the normal function of overlay network. For avoiding this problem, we proposed a Pull Model Clone algorithm to complete the clone process.

Definition 5. *Basic Unit of Time: The period during one hop in overlay network, to be measured with millisecond. We denote Basic Unit of Time by T, and T is determined by the status of Chord which CNChord is based on.*

Definition 6. *Current Idle Time: Current Idle Time is presented by Tnoact. One node's Tnoact is the period from the previous action finished to present.*

Definition 7. *Idle Trigger Threshold: Idle Trigger Threshold is presented by Tts. One node's Tts is the maximum period from the previous action finished to the next action occurs.*

When $Node_x$ is idle, we use Pull Model Based Clone algorithm to complete cloning $Node_s$. The following algorithm shows the detailed process. Clone($Node_s$, $Node_x$) procedure clones the parameter $Node_s$ to $Node_x$.

```
Algorithm 1. Pull Model Based Clone Algorithm Pull Clone (Node_s)
 {//passive clone algorithm on Node_x
if (Tnoact<Tts) then {waiting;}
else {Node_s=Find_successor(Node_x);
      if (!ISBusy(Node_s))
      then {Clone(Node_s, Node_x);}//clone Nodes to Node_x}
}
```

2.4 CNChord Routing Table Structure

The finger table of CNChord is composed of two finger tables with the same structure. They are expressed as FingerTable1 and FingerTable2. They are used

Table 1. Structure of FingerTable of CNChord.

Sequence	*Distance*	*Value*
0	$2^0 = 1$	$Node_x$
1	$2^1 = 2$	$Node_y$
2	$2^2 = 4$	$Node_z$
...	...	...

to store the routing information of Main Node and Clone Node. The structure of FingerTable1 and FingerTable2 are represent as Table 1.

Each finger table has m entities, which sequence is from 0 to $m - 1$ in increases of 1. m is the identifier length and it must be large enough to make the probability of two nodes or keys hashing to the same identifier negligible. The distance is 2 of i^{th} power, and i will increase from 1 to $m - 1$. $FingerTable1[i].value$ is the first node whose distance is equal to or greater than 2^i in clockwise from current node. Particularly, successor($Node_x$) = $Node_x.FingerTable1[0].value$.

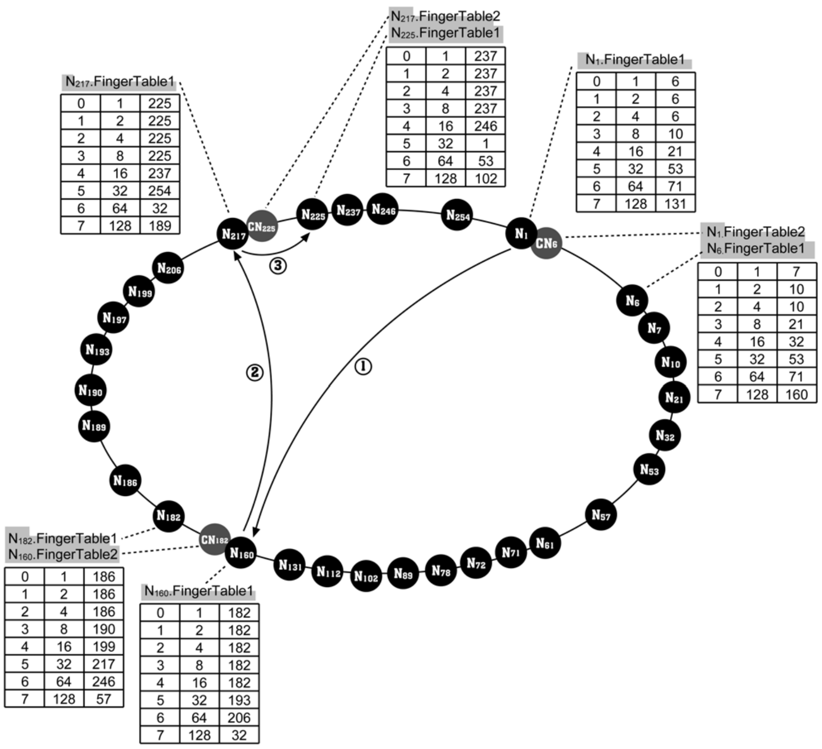
2.5 Query Mechanism

When $Node_x$ in CNChord received a query, The algorithm needs to determine whether id is between $Node_x$ and successor($Node_x$), if so the algorithm returns successor($Node_x$). If successor($Node_x$) has already been cloned to $Node_x$, then search in FingerTable2, else search in FingerTable1.

At the same time, we optimize the searching flow and propose an Optimal Backward Searching algorithm, which search the finger tables from end to begin, accelerating the speed of determining query range.

```
Algorithm 2. Optimal Backward Searching Algorithm
Node_x.find_clone_successor(id)
//Node_x search successor(id)
if (id is between (Node_x,successor(Node_x)]) then
    {Return successor(Node_x);}
else
    {
    Node_y=Node_x.closet_proceding_node(id);
    Return Node_y.find_clone_successor(id);
    }

Node_x.closet_proceding_node(id)
//Sub function description
for i=m downto 1 step -1 {
    if (FingerTable2 is NULL)
    then
    //Node_s is not cloned to Node_x,search in FingerTable1
        {
```


N217 CN225 N225 N237 N246 N254 N1 CN6 N6 N7 N10 N21 N32 N53 N57 N61 N71 N72 N78 N89 N102 N112 N131 N160 CN182 N182 N186 N189 N190 N193 N197 N199 N206
①
②
③

N217.FingerTable1

0	1	225
1	2	225
2	4	225
3	8	225
4	16	237
5	32	254
6	64	32
7	128	189

N217.FingerTable2
N225.FingerTable1

0	1	237
1	2	237
2	4	237
3	8	237
4	16	246
5	32	1
6	64	53
7	128	102

N1.FingerTable1

0	1	6
1	2	6
2	4	6
3	8	10
4	16	21
5	32	53
6	64	71
7	128	131

N1.FingerTable2
N6.FingerTable1

0	1	7
1	2	10
2	4	10
3	8	21
4	16	32
5	32	53
6	64	71
7	128	160

N182.FingerTable1
N160.FingerTable2

0	1	186
1	2	186
2	4	186
3	8	190
4	16	199
5	32	217
6	64	246
7	128	57

N160.FingerTable1

0	1	182
1	2	182
2	4	182
3	8	182
4	16	182
5	32	193
6	64	206
7	128	32

Fig. 2. Process of $Node_1$ search key 224 in a typical CNChord with 256 nodes

```
        if (Node_x.FingerTable1[i].value is between (Node_x,id))
        then
            {Return Node_x.FingerTable1[i].value;}
        }
    else
    //Node_s has been cloned to Node_x,search in FingerTable2
        {
        if (Node_x.FingerTable2[i].value is between (Node_x,id))
        Return Node_x.FingerTable2[i].value;
        }
} Return;
```

Figure 2 shows a typical CNChord with 256 nodes, and the process of $Node_1$ search key 224. The process of $Node_1$ search key 224 in Figure 2 is illustrated as follow:

(1)$Node_1$ queries the location of key 224, for successor(N_1)=N_6, $224 \notin (1, 6]$ and $N_1.FingerTable2 \neq null$,the algorithm lookups the key 224 in N_1.

FingerTable2 from the bottom and retrieved $N_1.FingerTalbe2[7].value = N_{160}$. Because $160 < 224$, the request is forwarded to N_{160};

(2)When $Node_{160}$ receives the request, for $successor(Node_{160}){=}Node_{160}.FingerTable1[0].value{=}N_{182}$, $224 \notin (160, 182] and N_{160}.FingerTable2 \neq null$, the algorithm lookups the key 224 in $Node_{160}.FingerTable2$ from the bottom and retrieves $N_{160}.FingerTable2[5].value = 217$.Here 217 is still smaller than 224, the request is forwarded to $Node_{217}$;

(3)When $Node_{217}$ gets the request, for $successor(N_{217}) = N_{217}.FingerTable1[0].value = N_{225}$, the algorithm successfully finds the key 224 and $224 \in (217, 225]$ returns the result to $Node_1$;

2.6 Node Joining

Node joining process in CNChord is very simple, and this can be divided into two sections: node joining into ring and node cloned.

Section 1: Node joins in CNChord ring. Before a node joins in the ring, it needs to be known at least one node which is already in the ring. Then new node can joins following the steps which are described in [1]. If the node is the first one of CNChord, then following the function create() described in [1].

Section 2: After a node joins the Chord ring, its predecessor adds the node as a successor. When the predecessor is idle, it can clone the new joined node according to the algorithm 1.

2.7 Difference Push Updating Algorithm

When the FingerTable1 of Main Node is changed, how to make the Clone Node changes its corresponding FingerTable2? To decrease the query message caused by pull mechanism, we propose a Difference Push Updating Algorithm.

```
Algorithm 3. Difference Push Updating Algorithm Push sync
(Node_s,Node_x)
//synchronize Node_s to Node_x
While (Node_s.FingerTable1[i].value changed) {
    Wait until (!ISBusy(Node_s))
        {
        Send(Node_s.FingerTable1[i], Node_x);
        Node_x.FingerTable2[i].value=Nodes.FingerTable1[i].value
        }
}
```

Function send() in algorithm 3 only transfer records which FingerTable1 are changed from Main Node to Clone Node. For example, in Figure 2 when $Node_{21}$ leaves CNChord, $Node_6$.FingerTable1[3].value will change from 21 to 32. System perform sync($Node_6$,$Node_1$) function which makes $Node_1.FingerTable2[3].value = 32$.

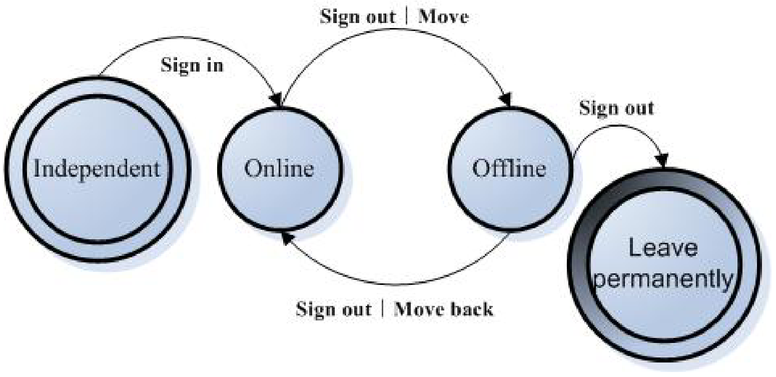


Fig. 3. State transition diagram of the process

2.8 Node Departing

Existing researches reflect the user's activities showing a very strong cyclical with the day [11, 12]. Node being steady means its activities in different days have many similarities, and we can foretell its future behavior by its resent behavior.

Let us consider a typical user (say $Node_A$)'s behavior in a P2P file-sharing system. $Node_A$ participants system in everyday's work time, i.e. $Node_A$ sign in at 9 o'clock when he start work and sign out at 17 o'clock when he leaves office. If the structure of this file-sharing system is Chord ring and $Node_B$ is the successor of $Node_A$. When $Node_A$ leaves the system, $Node_B$ will take part of $Node_A$, and the metadata stored in $Node_A$ should be moved to $Node_B$. The next day, when $Node_A$ sign in the system again, $Node_A$ will be treated as a new node, system have to perform the join process again, then the metadata yesterday stored in $Node_A$ and now in $Node_B$ should be moved back to $Node_A$. The state transition diagram of this process is shown in Figure 3.

The process described in Figure 3 will be repeatedly performed everyday. If there are a mount of nodes frequently change their states, Chord will produce a lot of maintenance messages to maintain its ring structure. We propose an Optimal Maintenance algorithm in CNChord to solve the problem that maintenance message is overhead in Chord when nodes frequently change their states.

Algorithm 4: Optimal Maintenance Algorithm

(1) In CNChord, when $Node_A$ leaves system, $CNode_A$ will take part of $Node_A$. Messages that be forwarded to $Node_A$ formerly now will be forwarded to $CNode_A$. For the rest of nodes in CNChord system, this process is transparent, they can work as if $Node_A$ is still stay in the system. Once $CNode_A$ is work, it will establish a log file to record all the changes. Furthermore, it is only needed to record the sequence which value is changed. The benefit of this mechanism is in(3), system only needs to transfer the last state to Main Node, which can decrease the log length and the number of message to be transferred.

(2) After $Node_A$ leaves system, its metadata still to be stored on $Node_A$, for the purpose of reusing at next time. System provides a mechanical for delete the message by user itself when needed, such as Node leaves system permanently.

(3) When $Node_A$ join CNChord again, it should be followed the steps described in Optimaintenance() function which copy $CNode_A$'s change from log to $Node_A$.

We can see that Optimal Maintenance algorithm needs only few maintenance messages even if node join and leave system frequently.

3 Bidirectional CNChord

We improve the search algorithm by adding some heuristic information and give the following definitions.

Definition 8. *DistanceThe distance between two nodes has two values, clockwise distance and anticlockwise distance. For example there are two nodes A with label a and B with label b, the clockwise distance between A and B is* $\mid b - a \mid$ *while the anticlockwise distance between them is* $\mid 2^m - (b - a) \mid$*.The distance with a minus in front, that means anticlockwise distance.*

Definition 9. *Midpoint Suppose that a node* N_1 *sends a request and we take* N_1 *as a center, the location that is* 2^{m-1} *to* N_1 *is called Midpoint.*

Definition 10. *Clockwise Midring and Anticlockwise Midring. The part between N1 and midpoint is called clockwise midring while the rest of the ring is called anticlockwise midring.*

Continuing our further research, since CNChord uses the similar clockwise search algorithm with Chord, CNChord inherits the defects belongs to Chord, that is the search algorithm always lookups the key clockwise not depending the location of the key on the Chord ring. The search algorithm always lookups along the clockwise, across midpoint and finally reached anticlockwise midring and reach on it. We provide a Bidirectional Clone Node Chord based on Clone Node Protocol (BCNChord). BCNChord use different search algorithms to lookups the key on the clockwise midring or anticlockwise midring according to the Distance between the requester and the responsor.

3.1 Anticlockwise Routing Table

Three finger tables constitute a BCNChord Node's routing table, FingerTable1 and FingerTable2 are introduced above. There is another finger table used for storing the information about anticlockwise searching, which is called anticlockwise route table, noted by FingerTable0. The structure of FingerTable0 is as follows.

There are some differences among FingerTable0, FingerTable1 and Finger Table2. In FingerTable0, Distance field's value is negative, means the Distance is an anticlockwise distance. The value field of the i^{th} row keeps the node which

Table 2. Structure of anticlockwise route table/FingerTable0

Sequence	*Distance*	*Value*
0	$-2^0 = -1$	$Node_a$
1	$-2^1 = -2$	$Node_b$
2	$-2^2 = -4$	$Node_c$
...	...	...

is the most distant from the query node along anticlockwise ring in the range of 2^i.Note that $Node_x$'s predecessor is not the value field in the 0th row, that is$predecessor(Node_x) \neq Node_x.FingerTable0[0].value$.

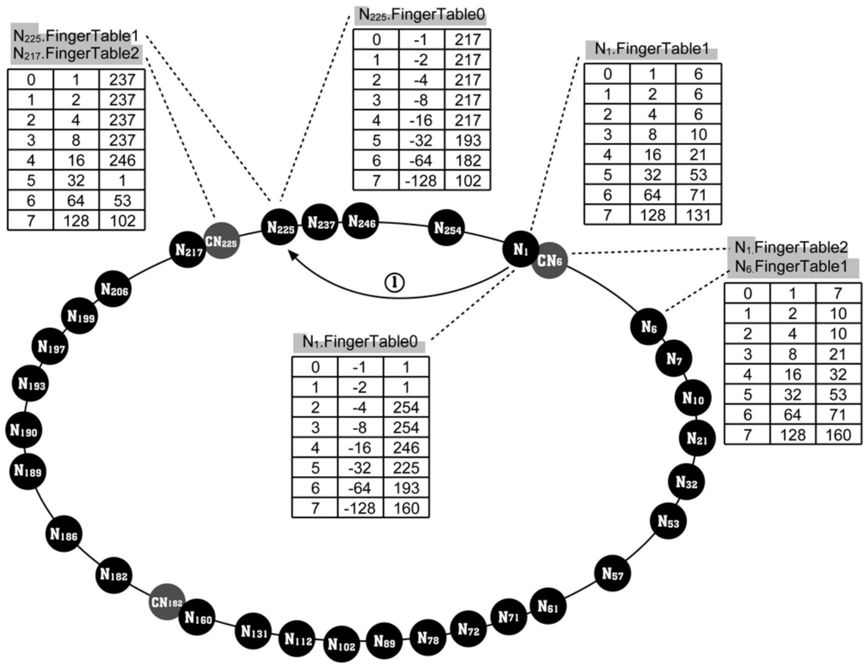
3.2 Query Mechanism

When a node begins to search a key, first the node needs to determine the relationship with the key. If the key is on the clockwise midring, the algorithm uses CNChord query mechanism, otherwise we propose an Anticlockwise Searching algorithm as follows.

```
Algorithm 5. Anticlockwise Searching Algorithm
Node_x.antisearch(id)
//Node_x search successor(id) anticlockwise
if (Node_x.FingerTable0 is NULL)
then{
//can not perform anti search procedure, change the search direction
    Node_x.find_clone_successor(id);}
else//perform anti search { if (id is between
(Node_x.FingerTable0[0].value,Node_x) then
    {Return Node_x.FingerTable0[0].value;}
else {
    Node_Y=Node_x.anticloset_proceding_node(id);
    Return Node_Y.antisearch(id);
} }
Node_x.antiproceding_node(id)
//Sub function description
{ for i = m downto 1 step -1 { if id is between
(Node_x.FingerTable0[i].value, Node_x.FingerTable0[i-1]]
    {Return Node_x.FingerTable0[i-1];
break;} } Return Node_x; )
```

Figure 4 shows the process of $Node_1$ querying key 224 on BCNChord, the process in CNChord is illustrated in Figure 2.

(1)In Figure 4, N_1 send a request, the algorithm begins to lookup key 224 in N_1's FingerTable0, since $N_1.FingerTable0 = N_1$, we need to lookups the key bigger than and most close to 224, $Node_{225}$ is found and the request is forward to $Node_{225}$.



N225.FingerTable1
N217.FingerTable2

0	1	237
1	2	237
2	4	237
3	8	237
4	16	246
5	32	1
6	64	53
7	128	102

N225.FingerTable0

0	-1	217
1	-2	217
2	-4	217
3	-8	217
4	-16	217
5	-32	193
6	-64	182
7	-128	102

N1.FingerTable1

0	1	6
1	2	6
2	4	6
3	8	10
4	16	21
5	32	53
6	64	71
7	128	131

N1.FingerTable2
N6.FingerTable1

0	1	7
1	2	10
2	4	10
3	8	21
4	16	32
5	32	53
6	64	71
7	128	160

N1.FingerTable0

0	-1	1
1	-2	1
2	-4	254
3	-8	254
4	-16	246
5	-32	225
6	-64	193
7	-128	160

Fig. 4. Process of Node1 search key 224 in a typical BCNChord with 256 nodes

(2)$Node_{225}$ receive the request and begin to lookup its FingerTable0, since $N_{225}.FingerTable0[0].value = N_{217}$ and $224 \in (217, 225]$,the algorithm retrieves $Node_{225}$ and returns the result.

As for the case in Figure 2, BCNChord only needs one step to reach destination node while CNChord needs three steps. Query efficiency in BCNChord greatly improved.

4 Performance Analysis

This section we do some analysis on CNChord's performance and BCNChord has the same performance.

4.1 Scarce Item Searching

There is a phenomenon that resources may be there but system can't find it in Chord. In Chord based structured P2P file-sharing system, each node decides what resources are saved, and stored (NodeID,ObjID) on the node whose ID is successor(ObjID). For example, globally unique resource Obj_1 is stored on $Node_1$, after the hash function, system store $(Node_1, Obj_1)$ on the $Node_{57}$, which is the successor of Obj_1. $Node_{57}$ leaves system at some time, and in a short period

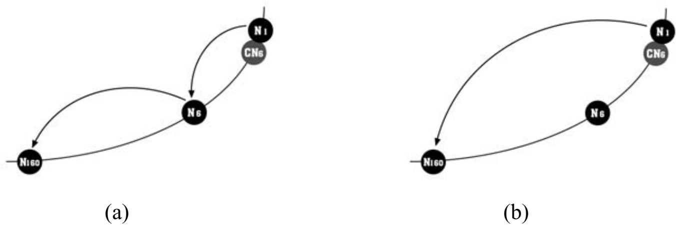


Fig. 5. (a) Search procedure in Chord. (b) Search procedure in CNChord.

system can't immediately update the finger table to reflect the change. Right now, there is a query about Obj_1, and the query will be failure although $Node_1$ who own Obj_1 is still in the system.

When CNChord is in the same case, since $Node_{57}$ is failure, whether the system has detected exception, and whether the finger table has been repaired, the message originally rooting to $CNode_{57}$ will directly rooting to $CNode_{57}$ now, and finally return the right result $(Node_1, Obj_1)$.

From the analysis above we can see that CNChord performs better than Chord in scarce item finding. Once the object is in CNChord, even if it had been in any node of CNChord, it certainly will be found.

4.2 Query Complexity

As for a Chord System having n nodes, the degree of Node in Chord is O(logn). Although CNChord has added Clone Node, the Clone Node has the same operational mechanisms with Main Node. We can draw the conclusion that the degree of Node in CNChord is also O(logn).

Theorem 1. *In an N-node CNChord network, a lookup requires at least 1/2O (logn) hops.*

Proof. First of all, in an N-node Chord network, each node maintains information about only O(logn) other nodes, and a lookup requires O(logn) hops[1]. In CNChord, each hop can achieve two hops in Chord. Figure 5 separately illustrates the procedure of $Node_1$ search key 160 in Chord and CNChord.

Figure 5 illustrates that $Node_1$ can search both at itself and $CNode_6$ in CNChord system, this procedure need two hops in Chord, i.e. two hops in Chord is the same with one hop in CNChord , so a lookup in CNChord requires at least $1/2O(logn)$ hops.

5 Experimental Results and Analysis

5.1 Experiment Environment

We use the Java to achieve a complete simulation of the Chord system, and on this basis, constructed a CNChord system and a BCNChord system.

Experiments are divided into five groups, the number of nodes in each group is N, N respectively for 2^8, 2^{10}, 2^{12}, 2^{14} and 2^{16}. The total resource number is 10^6. Resource number in each group is 10 multiple N, randomly selected from the total resource. Each group experiment is repeated 20 times, calculating the average value to be the result of this group. For each group, once the resources are selected, they'll not change until next group. Resources are allocated to nodes randomly. The number of resources one node can stored is from 0 to 100.

5.2 Initial Maintenance Cost

Let the system has zero node at the beginning, then N nodes randomly join the system with period T. T must be large enough to satisfy the period Chord and CNChord required to achieve stability. Supposing once a node joins the system, it will not leaves. This can be use of to simulate the creation of them. After the system state is steady, the number of maintenance message is showed in Figure 6.

Figure 6 shows the number of CNChord maintenance message is almost twice of Chord. It is because that during the establishment of CNChord, not only need to establish a Chord ring, but also need to clone all nodes to their predecessor. The message is almost doubled. We can see in Fig 8 the initial maintenance cost of BCNChord is more than CNChord because BCNChord has to maintain a reverse routing table but CNChord doesn't. Thus according to the initial maintenance cost of building system, Chord is the smallest, CNChord's is medium and BCNChord's is the largest.

5.3 Maintenance Cost of Node Departing

Then we let Main Node randomly depart the system in each group, and don't allow nodes join in the system during the experiment. Main Node department reason is

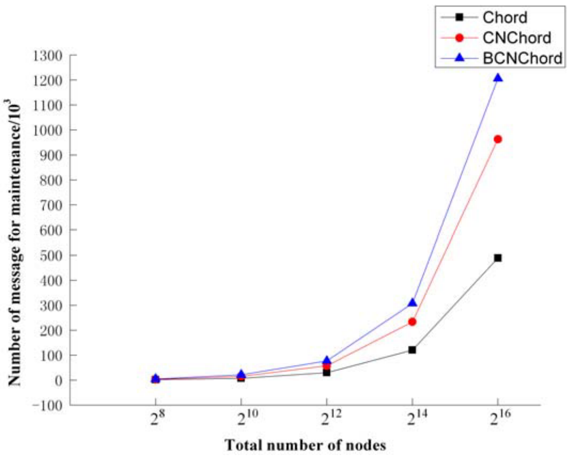


Fig. 6. Number of maintenance message for initialization

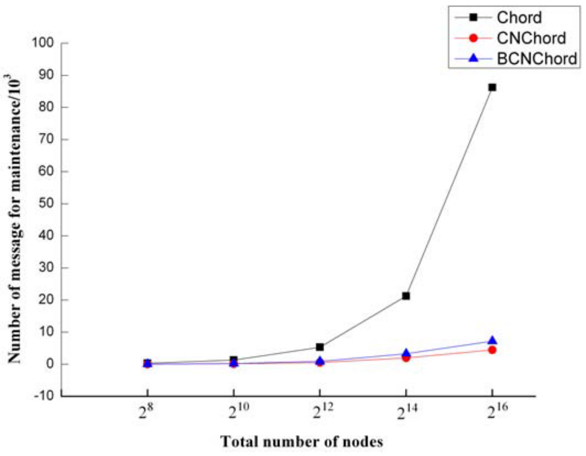


Fig. 7. Number of maintenance message caused by departing

divided into initiated leave and failure. The difference between them is that initiated leave Main Node will send a inform message to Clone Node, but the other don't send. The number of maintenance message is showed in Figure 7.

We can see that the number of maintenance message in Chord is far greater than CNChord. This is because when a node quits Chord system, the metadata stored on it must be moved to its successor, but CNChord needn't do this. The only thing CNChord need to do is modified finger table items which are changed. Under the circumstances where node only quits the system without joining, this kind of message almost can be ignored. Similarly, the maintenance cost of BCNChord is more of a reverse routing table compared to CNChord, this part can be negligible compared to Chord. The maintenance cost of BCNChord and CNChord is very close to such an extent that we can not distinguish them clearly in Figure 7 in the first several sets of experiments.

5.4 Maintenance Cost of Node Frequently Joining or Departing

In this experiment, there are 2^{12} nodes in the system. The experiment includes 5 groups, T respectively is 0.2 hour, 1 hour, 6 hour, 12 hour, 24 hour and 32 hour. In each group, 10 percent of nodes randomly depart and rejoin in the system once. The maintenance of message is shown in Figure 8.

From Figure 8 we can see that the maintenance cost curve of Chord decreases as frequency reduce. At the beginning, the curve decreases obviously, and the slope of the curve decreases as frequency reduce. It's because as the join and depart frequency reduce, the number of maintenance operations also decrease, so the impact on the system will be smaller.

At the same time, the percentage of maintenance cost of CNChord vs. Chord is become larger and larger, i.e. the optimal efficiency of CNChord to Chord decreased as frequency reduces. The percentage increase from 68.5% at 0.2 hour

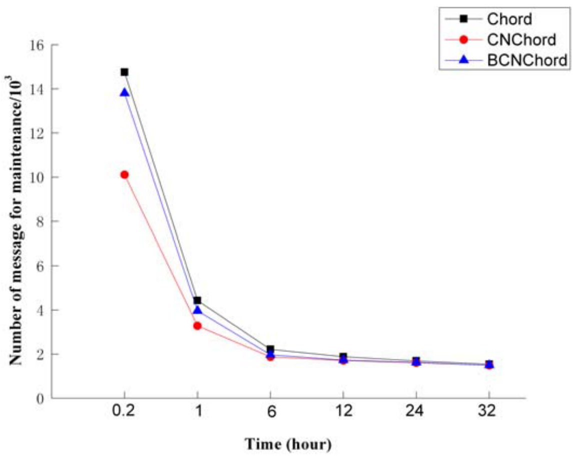


Fig. 8. Number of message caused by node frequently joining or departing

to 96.5% at 32 hour. In other words, the optimal efficiency decreases from 31.5% at 0.2 hour to 3.5% at 32 hour. We can draw the conclusion that the more frequent the node is, the better optimal efficiency of CNChord is.

The maintenance cost of BCNChord is very close to Chord, so we can't see the difference obviously in figure 8 . This is because the maintenance mechanism of reverse BCNChord routing table is very similar to Chord, both need to use much bandwidth.

5.5 Query Efficiency

Finally,in the case of 2^{12} nodes in the system,when the system is stable, we send 5 groups of request, each group contains 50 random request. The random request

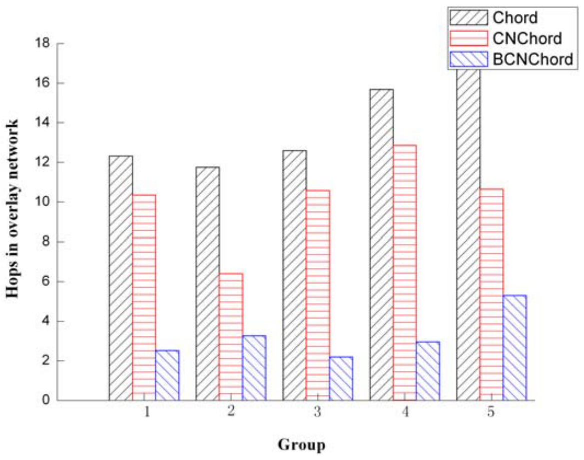


Fig. 9. Average hops in Chord, CNChord and BCNChord

is generated only in the Chord random test and kept in files. In every group test of CNChord and BCNChord, we use the same requests in a group , calculate the average value of response time and use the average value as this group's response time. Figure 9 shows the logical hops of Chord, CNChord and BCNChord.

It is can be seen that the use of a bidirectional mechanism, the query hop on logic layer of BCNChord is stably under 6, it is obviously better than the other two. When a node sends a request, BCNP chooses the shortest distance from the target node to initiate the direction of inquiry, and the extension of Chord and CNChord inquiries only clockwise direction. We can see that, CNChord is not stable contrast to the advantage of Chord, the best performance is about 1/2 , most of the situation is around in this range. This is because CNChord and Chord uses a similar search mechanism.

6 Conclusion

We propose a Clone Node Protocol to reduce the maintenance cost of structured P2P system. In order to verify the efficiency of CNP, we achieve a CNChord. Furthermore, we implement a BCNChord in order to increase query efficiency of CNChord. Theoretical analysis and experimental results show that CNChord can greatly reduce the cost of maintaining the P2P structure especially in the case of highly dynamic system and BCNChord can effectively improve the query speed, more suitable for real-time applications. But the latter one has the similar maintenance cost with Chord. In a word, CNP can effectively reduce the maintenance cost of structured P2P, and we can decide use CNChord or BCNChord by the requirements of different applications.

Acknowledgments. This paper is partially supported by Beijing Key Discipline Program.

References

1. Stoica, I., Morris, R., Karger, D., et al.: Chord: a scalable peer-to-peer lookup service for Internet applications. In: Proceedings of ACM SIGCOMM 2001, vol. 31, p. 149. ACM, San Diego (2001)
2. Karger, D.R., Ruhl, M.: Diminished Chord: A protocol for heterogeneous subgroup formation in peer-to-peer networks. In: Voelker, G.M., Shenker, S. (eds.) IPTPS 2004. LNCS, vol. 3279, pp. 288–297. Springer, Heidelberg (2005)
3. Montresor, A., Jelasityt, M., Babaoglu, O.: Chord on demand, vol. 2005, pp. 87–94. IEEE Computer Society, Los Alamitos (2005)
4. Zols, S., Schollmeier, R., Kellerer, W., et al.: The hybrid chord protocol: a peer-to-peer lookup service for context-aware mobile applications, vol. 2, 12, p. 12. Springer, Berlin (2005)
5. Yuh-Jzer, J., Jiaw-Chang, W.: Chord2: a two-layer Chord for reducing maintenance overhead via heterogeneity. Computer Networks 51(3), 712–731 (2007)
6. Xue, K., Hong, P., Li, J.: A new P2P model with fractional steps joining. Guadelope, French southern territories: Institute of Electrical and Electronics Engineers Computer Society 2006, 98 (2006)

7. Leong, B., Liskov, B., Demaine, E.D.: EpiChord: Parallelizing the Chord lookup algorithm with reactive routing state management. Computer Communications 29(9), 1243–1259 (2006)
8. Chen, D., Yang, S., Peng, X.: TaChord: A Chord system using topology-aware routing and super peers. Journal of Southeast University 20(3), 273–278 (2004)
9. Yunhao, L., Xiaomei, L., Li, X., et al.: Location-aware topology matching in P2P systems, vol. 4, pp. 2220–2230. IEEE, Piscataway (2004)
10. Feng, H., Minglu, L., Minyou, W., et al.: PChord: improvement on Chord to achieve better routing efficiency by exploiting proximity. IEICE Transactions on Information and Systems E89-D(2), 546–554 (2006)
11. Tati, K., Voelker, G.M.: On object maintenance in peer-to-peer systems (2006)
12. Liu, H.Y.: Analysis of resource characteristics and user behavior in P2P file sharing system maze [MS. Thesis] (2005)