On the Convergence of Chord Network

NIE Xiao-wen LU Xian-liang DUAN Han-cong LI Lin PU Xun
School of Computer Science & Technology, UESTC, ChengDu 610054, China
{niexiaowen, xlu, duanhancong, lilin, puxun}@uestc.edu.cn

Abstract

Due to the churn, the P2P overlay network can not be static. The dynamic environment may degrade the services provided by Distributed Hash Table (DHT), then the convergence of the overlay is very important. In this paper, we try to analyze the convergence of Chord network. We prove that the generic structures of Chord network is a pseudo-tree, which can be converged to a circle by algorithm stabilize. The ring in Chord has more meanings than a circle in Graph theory, and the ring requires all nodes in the network permuted by their ids. An efficient strong_stabilize algorithm is proposed to make the network converge to a ring rapidly.

1. Introduction

The rising of P2P technology in recent years has drawn the attention of many researchers, and the new technology has been successfully applied in a lot of downloading tools. As a new member of P2P technology, Distributed Hash Table (DHT) is known for its free scaling and determinate searching. But the advantages of DHT are gained by the correct topology of overlay.

On the other hand, due to the churn, nodes ceaselessly join and depart from the network, so the topology of the network can not be supposed as a static structure. The dynamic topology may impact the retrieving of searching in the network. Many researchers have investigated the problem of churn in DHT. They[1] or quantitatively analyze the relationship between retrieving rate and the churn, or[2] propose algorithms to resist the churn. In this paper, we try to study this problem from another aspect. Our focus is put on the convergence and the maintenance protocol of the topology.

We choose Chord[3] as the sample of DHT for its simplify and fault-tolerance. We will try to understand the topologic shape of the overlay under churn, and analyze the how the stabilize protocol can converge the overlay to the supposed topology. At last, we put forward a new strong-stabilize algorithm which can find out rapidly whether the nodes in the network are permuted with wrong order.

2. Review of Chord

Before the analysis, we would like to introduce some terms appeared in this paper. Let the online nodes set in Chord network to be *N*, and each node is assigned with an identifier (*ID*), which is a fixed-length binary string. Object or file stored in the nodes acquires the key by computing the SHA value with name or content. The length of key and ID is equal to *m*. The online nodes spread in the id space sparsely. Therefore, each node has to guard a region of ID and store the keys of objects which belong to the region. The *guard* region is defined as the segment of ID space from the previous node to the node itself, which is denoted as $(u.pre,u]$, where *u.pre* means the previous node of node *u*.

The most import data structure of every node in Chord is *succeeding pointer* which points to the next online node whose ID is more than the current node. And *preceding pointer* points its predecessor whose ID is less than the current node. With succeeding pointer, all nodes constitute a clock-wised ring. To search an object or a node, the messages can be sent along the ring, then eventually reach their destinations. But this hop by hop method is too slow, Chord use *fingers* table to accelerate the routing process. Fingers table is an array sizing *m*, and each item in fingers table points to the succeeding node of $id = u + 2^{i-1}$. When one node forwards a message, the node will use the maximum item which is less than the destination in fingers table as the next hop of the message.

As a result, Chord decomposes the routing algorithm into two levels: correctness and performance. It first implements a hop by hop consistent algorithm in bottom and then introduces an efficient greedy routing algorithm in higher level. When the greedy algorithm of upper layer runs abnormally, Chord can automatically switch to the consistent algorithm at bottom. Although succeeding pointers and fingers table are all used in the routing algorithm, since the ring topology is formed with succeeding pointers, we omit the fingers table in the following analysis.

3. The Topology of Chord

The routing algorithm in Chord assumes that the topology of the overlay is a ring, which means that all nodes are permuted orderly. But under churn, this ideal topology is difficult to maintain, because in a distributed computing environment, it is impossible to keep all distributed states consistent with fault-tolerance[4]. There is no algorithm could guarantee that all succeeding pointers point to correct nodes under churn.

If a succeeding pointer is invalid, we denote it as ***nil***, and the out-degree of the node will be 0 at this time. Since there is only one succeeding pointer for every node, the out-degree of a node is either 1 or 0.

Lemma 1 *If the out-degree of all nodes is* 1*, there exists one circle in the network.*

Proof: Define closure $S(u) = \{v \in N \mid v = u.suc^i; i = 0,1,...\}$, and let $S(u)^{i+1} = \{S(u)^i, u.suc^{i+1}\}$.

$S(u)^0 = \{u\}$

$S(u)^1 = \{S(u)^0, u\}$

...

$S(u)^i = \{S(u)^{i-1}, u.suc^i\}$

If $u.suc^i \in S(u)^{i-1}$, then $S(u)^i = S(u)^{i-1}$, which denotes $u.suc^i = u.suc^j, i \neq j$. Let $v = u.suc^j$, then

$S(v)^0 = \{v\}$

$S(v)^1 = \{v, v.suc^1\}$

...

$S(v)^{i-j} = \{S(v)^{i-j-1}, v.suc^{i-j}\}$

Since $v = v.suc^{i-j}$, $|S(v)| = i - j + 1$, then there exists one circle $\{v, v.suc, ..., v.suc^{i-j}\}$. □

Definition 1 (Split Network) *If* $\exists u_1, u_2 \in N : u_1 \neq u_2$, $S(u_1) \cap S(u_2) = \Phi$*, we call the network split. And if* $\exists u_1, u_2, ... u_k \in N : u_1 \neq u_2, u_1 \neq u_3, ..., u_{k-1} \neq u_k, S(u_1) \cap S(u_2)$, $,..., S(u_{k-1}) \cap S(u_k) = \Phi$*, i.e., there exist k nodes that are unequal to one another, and their closures do not intersect each other, we call the network is split into k subnets. If all succeeding pointers are valid and the network is not split, the network is called connected.*

Theorem 2 *If the network is connected, there exists one and only one circle.*

Proof: Assume by contradiction that there exist two or more circles in the network, denoting the two circles as $S(u)$ and $S(v)$. If $u = v$, because the out-degree of all nodes is 1, then $S(u) = S(v)$, $u \neq v$. If $S(u) \cap S(v) \neq \Phi$, we can choose an intersected node in $S(u) \cap S(v)$, then $S(u) = S(v)$. So $u_1 \neq u_2 \wedge S(u_1) \cap S(u_2) = \Phi$, but the network is connected, that is a contradiction. □

There exists only one circle in a connected network, but this does not mean all active nodes are on the circle. The succeeding pointers of many nodes may directly or indirectly point to the circle.

Lemma 3 *In a connected network, the nodes not on the circle constitute a forest.*

Proof: If the arcs between the nodes on the circle are taken away, there is no circle existing according to Theorem 2; so all nodes of the network constitute a forest according to the definition in Graph theory. Since the out-degree of a node is 1, the nodes on the circle are the roots of these trees. □

From Lemma 3, if the whole circle is viewed as a virtual node, the forest forms a tree, so the dynamic topology of Chord is called *pseudo-tree*. The ring is the ideal topology after Chord network has been converged. However, the frequent joining and leaving of nodes will make this ideal status difficult to retain. The pseudo-tree is a more generic snapshot of Chord network structure.

4. Convergence to a Circle

To converge the pseudo-tree to a circle, every node must run stabilize protocol. Algorithm 1 is the pseudo-code of stabilize, which spends less time on communication than the original algorithm[3].

```
u.stabilize()
  suc=suc.notify(u);

u.notify(v)
  if ( $pre = nil$ or $v \in (pre, u)$ )
    pre = v;
      retrun u;
    else
      return pre;
```

Algorithm 1 *Stabilize* algorithm

Definition 2 (Convergent Subtree) *Assume u is one node on the circle of a connected Chord network N,* S(*u*) *denotes the circle, and* $N - S(u)$ *denotes the set of all of the nodes which are not on the circle. Define* $\exists i > 0 : \forall 0 < j < i : T(u) = \{v \in N - S(u) \mid v.sub^i = u \wedge v.sub^j$

$\notin (S(u)-u)\}$, *where T(u) is the subtree whose root is u. If* $\forall v \in T(u): v < v.suc \wedge v > u.pre$ *, we call T(u) as convergent subtree.*

In Definition 2, T(u) is a subtree whose root is a node u on the circle. If any node in the T(u) satisfy that it is less than the predecessor of the root of T(u) on the circle and its successor is greater than itself, then the substree is convergent.

Lemma 4 *In a connected network, algorithm stabilize converges the convergable subtrees onto the circle, and permutes the nodes locally.*

Proof: Assume *u* is the root of one covergable subtree *T*(*u*), and *v* is the preceding node of *u* on the circle. We extends *T*(*u*) to include *v*, let *v* to be a son node of *u*. The result of convergence of an extended subtree *T*(*u*) means that all notes in the extended subtree permute as a chain according to the order of *id*, and the head of the chain is *v*, while the tail of the chain is *u*.

Let $T(u)=\{w_1,w_2,...,w_k\}$, and $k=|T(u)|$, while $w_1 < w_2 < ... < w_k$, $u = w_k$ and $v = u.pre = w_1$.

1) Since $\forall w \in T(u): w < w.suc \wedge w < v$, so $w_{k-1}.suc = u$; running *stabilize* on w_{k-1} , will make $u.pre = w_{k-1}$. All the other son nodes of *u* runs *stabilize*, since $w_i < w_{k-1} = u.pre$, the preceding pointer of *u* will not be changed from w_{k-1} , furthermore the succeeding pointer of the other son nodes will point to w_{k-1} , $w_i.suc = w_{k-1}$.
2) Any node *w* in *T*(*u*), $\forall w \in T(u)$, running *stabilize*, will satisfy $w < w.suc \wedge w < v$. So $T(w_{k-1})$ is still an extended subtree. We continue the above process recursively, $T(u)=\{w_1,w_2,...,w_k\}$ will be converged to a chain.

Since the converging process is only done on the set T(u) , so the order of the chain is local, and the convergence of stabilize is local. □

After the stabilizing process, the nodes in the convergable subtree are permuted between the root node and the predecessor of the root note on the circle.

Theorem 5 *In a connected network, algorithm stabilize converges the nodes of all subtrees into the circle.*

Proof: Let the root of one subtree is *u*, and $u.pre = v$. We color up the convergable subtree *T*(*u*) of *u*'s subtree with blue, and the other part *R*(*u*) with red. These red nodes in *R*(*u*) are divided into two classes:

1) $\forall w \in R(u): w > u \vee w < v$. The succeeding pointers of this kind of nodes will point front of *v*, and search a proper node as its succeeding node hop by hop.
2) $\forall w \in R(u): w < u \wedge w > v$. If these nodes satisfy the condition of being in a convergable subtree in the stabilizing process of *T*(*u*), according to Lemma 4, they will be converged. Otherwise, they will converge like instance of 1). □

The stabilizing process of generic subtree is iterative. When one node in the subtree is converged onto the circle, then the new subtree whose root is the node begins a new stabilizing process. If in the stabilizing process, one node is greater than the root or less than the predecessor of the root on the circle, this node will have to searching its appropriate position hop by hop from the anti-clockwised direction. Because in this situation, the stabilizing process will be very slow, these nodes would better rejoin the network to find its proper successor.

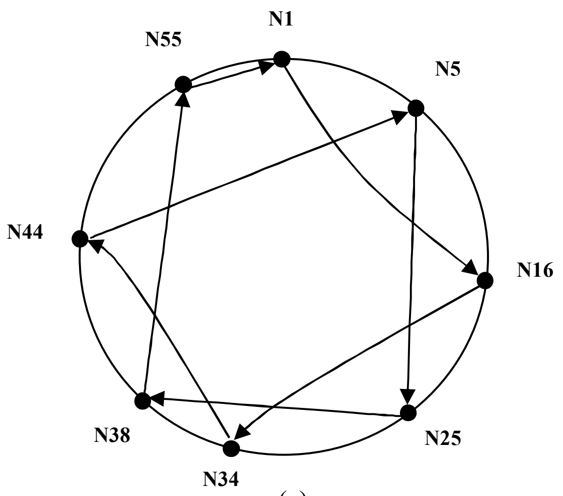
5. Convergence of Loopy Status

In the last section, we have proved that Algorithm 1 could stabilize the network onto a circle. But as we have pointed out, the ring in Chord have more meanings than a circle in Graph theory. Liben Nowell[5] has given an example in which the network converge to a circle but not a ring. As shown in Fig.1(a), the succeeding pointer of N16 points to N34, but the global next node of N16 is N25. At this time, running stabilize can not make the network converge to a ring. Liben Nowell calls this status loopy. A loopy network can be thought of as the circle nodes folded together, as shown in Fig.1(b).

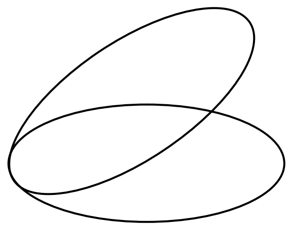
Definition 3 (Folding Number) *Suppose a network N is converged to a circle. For some id, if* $\exists u \in N: id \in (u, u.suc]$ *, we call a thread traverses id, and we call the number of threads traversing id the folding number, which is denoted as fold(id). If the network is connected, since there exists only one circle, so* $fold(id) \geq 1$.

Lemma 4 *In a loopy network which has converged to a circle, the folding number of any id is equal.*

Proof: Assume by contradiction that $\exists id_1, id_2 \in N: fold(id_1) > fold(id_2)$. Consider a range $[id_1, id_2]$, its indegree is $fold(id_1)$, and the outdegree is $fold(id_2)$. According to Theorem 2, the nodes in a thread are permuted as a chain, so each inputting thread of the range has one corresponding outputting thread. Since $fold(id_1) > fold(id_2)$, there exists a node in the range whose indegree is greater than 1. Because the network has been converged to a circle, the indegree and outdegree will be 1, contradiction. So the folding numbers of nodes are the same. □



(a)



(b)

Fig.1 The *loopy* status of Chord

To eliminate the loopy status, we need only to make the folding number of the network be 1. When some node finds that there exists folding of network, what the node needs to do is adjusting its succeeding pointer to point to a proper node. But how can a node detect the network entering loopy status? Stoica[3] gave an interesting method. He makes use of the characteristic of uni-directive searching in the Chord. A node searches itself on the circle, and if the result is greater than its id, then the network enters loopy status.

But the self-searching is hop by hop, that is too slow, and the complexity of this method is $O(N)$. Stoica argues that the probability of appearing loopy status in the network is very little, and the so called strong_stabilize algorithm can be run in a very long period in the background. But In our opinions, the robustness of the hop by hop searching can not be guaranteed in a dynamic environment. Why do not Stoica make use of the *fingers* table to accelerate the searching? Because the nodes in the *fingers* table may lie on the other thread, the self-searching may fail to detect whether the network is loopy.

To improve the efficiency of strong_stabilize algorithm, we modify the original fingers building algorithm[3], as shown in Algorithm 2.

Definition 4 (Succeeding Sections) *Suppose that the network N converges to a circle,* $\forall u \in N : u = u.suc^k$ *, where k denotes the number of nodes. From a node u in N, we divide the all of the succeeding nodes of u into fold(u) sections:* $[u, u.suc^{k_1}]$ *,* $[u.suc^{k_1+1}, u.suc^{k_2+1}]$ *, … ; where* $u.suc^{k_i} < u < u.suc^{k_{i+1}}$ *. We denote them as* $span_1(u)$ *,* $span_2(u)$ *, … .*

```
Algorithm 2 Build fingers table
/* build fingers table recursively */
u.build_fingers()
  $i_0 = \lfloor \log(suc - u) \rfloor + 1$;
  for ($i = i_0$) upto m
      $fingers[i] = fingers[i-1].find\_suc(u+2^{i-1})$;
```

Algorithm 2 building *fingers* table

The principle of Algorithm 2 is that the direct succeeding node of one node does locate in the first succeeding section, and if we recursively construct the fingers table from the direct succeeding node, then we can guarantee that all of the items of fingers table are located in the first succeeding section, which means that all the items of the table are in the same thread of the node.

Theorem 5 *With Algorithm 2, the fingers table of one node only contains the nodes in the first section.*

Proof: Suppose node $u \in N$. First we assume that the items of *fingers* increase with *id*, and $u < u.fingers[m]$, *m* denotes the binary length of ID.

Let $u.suc = u.fingers[1]$, so $u.fingers[1] \in span_1(u)$. $u.fingers[2]$ may be $u.suc$ or $u.suc.fingers[1]$. If $u.fingers[2] = u.suc.fingers[1] = u.suc^2$, since the items increase with *id*, which means $u < u.suc^2$, so $u.suc^2$ is in the first succeeding section, and $u.suc.fingers[1] \in span_1(u)$.

We recursively do the process above, $\forall u.fingers[i] \in span_1(u)$. □

```
Algorithm 3 Eliminate folding of network
/* build fingers table recursively */
u.strong_stabilize()
  v = u.suc.find_suc(u);
  if ( $u \ne v \wedge v < u.suc$ )
    u.suc = v;
```

Algorithm 3 *strong_stabilize* algorithm

Since the items of the fingers table are all in the first succeeding section, one node can safely use the fingers table to search itself. In Algorithm 3, if the returned value v of self-searching is not itself u, $u \ne v$, then

IEEE COMPUTER SOCIETY

there exists folding in the network. Obviously, the complexity of Algorithm 3 is $O(\log N)$.

6. Conclusions

In this paper, we choose Chord as a sample of DHT to analyze the convergence of its topology. The ideal topologic structure of Chord is a ring, but in a dynamic environment, the snapshot of the overlay is a pseudo-tree. The *stabilize* algorithm could converge the pseudo-tree to a circle. The ring in Chord means that all nodes are permuted in globally order, and occasionally the network may enter a loopy status. So nodes have to constantly check whether the network is loopy. To accelerate the detecting, we propose a new method on how to build the *fingers* table, which is recursively constructed from the first succeeding node of nodes. With *stabilize* and *strong-stabilize* algorithm, the network of Chord can converge to a ring.

7. References

[1] Condie, T., et al. “Induced Churn as Shelter from Routing Table Poisoning”. in *NDSS*. 2006. San Diego, California, USA: The Internet Society.

[2] Rhea, S., et al., “Handling Churn in a DHT”. 2003, EECS Department, University of California, Berkeley.

[3] Stoica, I., et al., “Chord: a scalable peer-to-peer lookup protocol for Internet applications”. *Networking, IEEE/ACM Transactions*, 2003. 11(1): p. 17-32.

[4] Lynch, N.A., *Distributed Algorithms*. 1996: Morgan Kaufmann Publishers Inc. 827.

[5] Liben-Nowell, D., H. Balakrishnan, and D. Karger, “Analysis of the evolution of peer-to-peer systems”, in *Proceedings of the twenty-first annual symposium on Principles of distributed computing*. 2002, ACM Press: Monterey, California.

IEEE COMPUTER SOCIETY