ISSN:0975-9646

Efficient Resource Allocation and Tracking in Structured P2P network: Back Trackable Finger Table based CHORD (BTFT-CHORD) Protocol

D.srivihari, P.Prasanna Murali Krishna

Dept of ECE, SGIT, Markapur , AP,India

Abstract: Recently, P2P (Peer-to-Peer) technology has witnessed a rapid development. Basically, one of key components in successful P2P applications is how to efficiently look up resources. Considering that structured P2P is a relatively efficient way to locate resources, this paper conducted two improvements to increase the search efficiency in Chord-based algorithms, one of the most popular structured P2P resource lookup protocols. In detail, our contributions are twofold. First, considering the fact that routing information in Chord is not abundant enough for efficient resource search, and looking up resource can only be enforced in clockwise direction, a new algorithm called BTFT-CHORD is proposed to reconstruct the finger tables in Chord, in which counter-clockwise finger table is added to achieve resource queries in both directions, and the density of neighboring fingers is increased. Additionally, BTFT-CHORD implements a new operation to remove the redundant fingers introduced by adding fingers in BTFT-CHORD. Experimental results show that BTFT-CHORD's query efficiency has been improved in terms of the average lookup hops and average lookup delay. The proposed BTFT-CHORD algorithm enlarged the finger table which may cause the forwarding-storm of routing maintenance messages.

INTRODUCTION

Sometimes in peer to peer networks each node acts as a server along with a client. It means it may behave both as a source and a receiver. In recent usage, peer-to-peer has come to spell out applications in which consumers can make use of the Internet to exchange files with each other directly or via an intermediate server. In version, however, there is absolutely no centralized server, but rather an interconnected community of peers. Each node can request information or files from any other node at any other level in time. The concept of P2P networks is that there is no centralized server here, all the info as well as the information is stored in a distributed fashion on all nodes.

In ordered peer-to-peer networks, connections in the overlay are fixed. They typically use distributed hash table based (DHT) indexing, like in the Chord system (MIT). Unstructured peer-to-peer networks don't provide any algorithm for organization or optimization of network connections.

There are three versions of unstructured P2P which have been defined as of now. In the very first kind of unstructured P2P community, I.e. pure peer-to-peer systems the entire network contains just peers with equal potential. As there aren't any special or high-priority nodes of any special infrastructure function, hence there is just one routing level. Hybrid peer-to-peer systems permit such infrastructure nodes to exist; it means they allow unique nodes with high priority that are known as superb nodes. The third type of unstructured P2P systems are central peer - to - peer systems, where there is a central server can be used for indexing capabilities and also to weight and build the whole method of unstructured nodes. The very first prominent and popular peer - to - peer file-sharing method, Napster, was really an instance of the design. Freenet and Gnutella, in the flip side, are samples of the model. Kazaa is an instance of the cross model.

We'll mostly handle ordered P2P systems. Organized P2P network apply a globally consistent process to ensure that any node can search and route every other node that includes the needed file, even though the file is found at some distant point within the system. Such a guarantee requires a far more organized pattern of links containing path details. Distributed hash table (DHT) is a structured P2P system, when a version of consistent hashing is used to assign storage details of every file.

Distributed Hash Tables

Distributed hash tables are distributed systems that provide lookup into a hash table. Every hash table is composed of (key, value) set and any node can effortlessly obtain the value associated with a vital. Responsibility for maintaining the maps from keys to values is spread among the nodes, in such a manner that a change in the set of individuals causes a minor amount of disruption. DHTs could be used to construct huge sites as they're distributed evenly across all nodes. This at a way increases the amount of nodes which can be in a system.

LITERATURE STUDY

Chord protocol [1]

A major problem with peer to peer programs is the fact that they're not able to effortlessly find the nodes including a specific information item. To fix this chord protocol was developed which is really a distributed lookup protocol. Its primary purpose would be to chart a specified key onto a node. Note is actually a scalable process.

Chord offers fast distributed computation of a hash function maps keys to nodes associated with them. Chord utilizes consistent hashing to supply keys to nodes. Whenever a fresh node enters the setup the tips are evenly distributed to each of of the nodes thus keeping a nicely distributed load. Because a chord node stores info about a few of the other nodes located close to it, Therefore chord method is scalable. All of this information is saved in a dispersed manner, so each node receives the hash value from other nodes.

Presently the primary concern is that how can we distinctively recognize every node. This hashing is achieved utilizing a hash function. As described before the principal use of a hash function would be to produce an m-bit identifier. This procedure is recognized as consistent hashing. Consistent hashing achieves the function of assigning keys to the nodes in the subsequent manner. All the identifiers are set in an identifier circle modulo 2^m. Now the keys are allocate to the nodes. This is done by evaluating the identifier of a key with the identifier of a node. The key gets assigned to a exacting node whose identifier's value is additional than that of the key. The selected node will be termed the descendant of the assigned key. It is also signify as successor(k), which means descendant of key k. Assume that identifiers are symbolized in a circle of numbers initial from 0 to $2^m - 1$, then successor(k) will be the first node when we traverse in a clockwise direction from k.

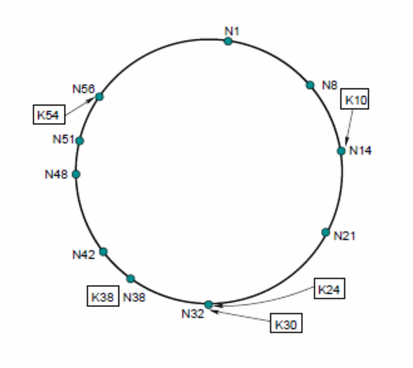


Figure1. An identifier circle (ring) consisting of 10 nodes storing five keys [1]

"Figure 1 shows a Chord ring with m = 6. The Chord ring has 10 nodes and supplies five keys. The successor of identifier 10 is node 14, so key 10 would be positioned at node14. Similarly, keys 24 and 30 would be positioned at node 32, key 38 at node 38, and key 54 at node 56" [1].

Chord lookup algorithm [1]

Typically in note protocol lookup is performed when each node tries to ask its successor about the key. Therefore it needs to continue bridging the chord ring one node after the other to look for the crucial. Today it is just a time-taking procedure. So there's a strategy to create the chord method scalable. Chord protocol maintains a finger table to look for the key in a more time efficient way Let us suppose that the number of bits in the identifier be m. A routing table is preserve by every node which contains an utmost of m entries. This table maintained by each node is called the finger table. The finger table enclose a number of entries, where the ith entry contains a mapping to the node's first descendant f which succeeds the node let us say n by at least $2^i - 1$ on the identifier circle, i.e., $f = \text{successor}(n+2^{i-1})$. We call node f the ith manipulate of node n. It is indicate as $n.\text{finger}[i]$. The finger table can be implicit more clearly from the figure shown below. Basically it has two entries, one for the identifier and the other for the IP Address of a meticulous node.

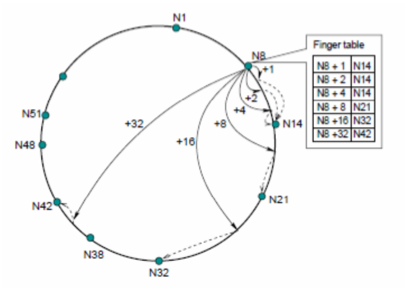


Figure2. Finger table entries for each node in chord ring [1]

The Figure 2 shows the finger table for node N8. The first finger of node 8 points to its descendant which is node 14, because node 14 is the first descendant which succeeds $(8+2^0) \bmod 2^6 = 9$. Similarly, the last finger of node 8 points to node 42, as node 42 is the first node which is the successor of $(8+2^5) \bmod 2^6 = 40$" [1]. It is apparent from the figure that the first finger for every node is its first successor.

More approaching can be gained on the finger tables from the table given below. It gives a obvious idea of the definition of fingers and the respective successors and predecessors for each node.

Finger Table definition [1]

finger[k]	**first node on circle that succeeds** *(n +*
successor	**the next node on the identifier circle;**
predecessor	**the previous node on the identifier circle**

The working of note protocol is quite straightforward. It demands another node to uncover a successor for it, whenever afresh node enters within the device. Whenever a brand new successor is located for it, it is subsequently given to it like its successor. Currently the successor also understands the cutting-edge node is its predecessor. Nevertheless, the preceding predecessor of the new node's heir doesn't know whether a different node as been put into the method. Because of this each node is immediately refreshed after a certain amount of time. During this node each node asks its forerunner and successor about any adjustments to know whether or not a new node has been added or removed from the existing system. That is the location where it comes to know that the setup has changed. Therefore the preceding predecessor of the recent node's successor is aware of its successor's new predecessor. Then it stores the nod as its heir. The new node that has been unaware of its own predecessor now realizes that it has as its predecessor this node. This is the way the device is managed. Every time a node fails or randomly leaves the device the keys stored on this node are evenly distributed to other nodes. But to preserve the sequence each node keeps independent table where it stores the records of next n nodes succeeding it or previous it. It keeps on sending messages in a consecutive order to every other node within the desk till it receives a reply from the node. The node which sends a reply to it first in order to what is stored within the entries is saved as its new heir and that node stores it as its heir.

Limitations in chord protocol

Because the existing method keeps a hand table when the search is done in a sequential manner within the clock wise direction there's an issue with all the time obtained to search a node. Consider a case where a trick to be researched is found on the node that is at the end of the ring when scanned in a clockwise path. This research would take plenty of time, thus cutting down the efficiency of the chord process. So it becomes quite important to minimize the sum of time required to research for a node in a less amount of time. That is a key feature which should be taken into consideration because the most important purpose of any protocol could be the quick and effective lookup of nodes containing keys.

OBJECTIVES

The undertaking is aimed at handling a number of the disadvantages within the first CHORD algorithm. Specifically the way it searches for a particular key within the NOTE ring. It only queries it in clock-wise direction. This single direction search not only raises time for finding the special key, but more routing communications must be passed in theP2P network. Seeking in both direction may decrease the time required to locate an unique important within the band. That is indeed more efficient if the key lies within the searching nodes predecessor. Within the regular formula the entire CHORD ring will be traversed, but in bidirectional look-up this will be faster and will require lesser hops.

DESIGN

This task tries to beat a number of the limitations with CHORD protocol mentioned in previous sections. The undertaking implements a bi-directional finger table, which would lessen the look-up time for a particular a node. In original NOTE all the messages are passed in a clockwise way over the ring , that is wasteful. As an example to lookup a central located near but previous the node, the lookup communications will have to navigate almost the whole CHORD ring.

At the core of the project is to implement the zero finger table for each node in addition to this already present finger table. Merely stated the finger table stores the successors and their mapped keys for a specific node, the anti finger table will have the list of predecessor nodes of that particular node. As finger table links to nodes in clockwise direction, the zero finger table will hyperlink the nodes in direction. Like whenever there is a link in hand table of initial NOTE form node A to node W, we add a reverse hyperlink from node W to node A. These change links which are in anticlockwise direction form a reverse hand table for this node. Having stated this there should be no change in how the data objects are saved. All of the data objects continue to be found at the successor of these keys.

So each node in modified CHORD protocol maintains:

- Finger Table
- Successor List
- Anti finger Table

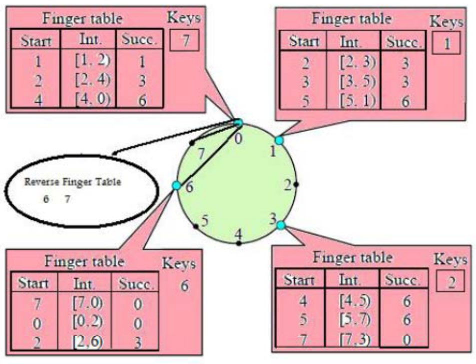
While the previous two are the same as they were in the original CHORD protocol.

Lookup Algorithm

Once we need to maintain the key and data mappings intact the research algorithm requires to be changed. The manner first NOTE functions is that it really searches for a particular key's successor. But when using the anti-finger stand, we store the node's forerunners so we are unable to use the same formula to really get the keys. Whenever we're using anti hand table, the keys won't be in the keys heir but its precursor. This is relevant to maintain the keydata mappings of the first method.

Using antifinger in a few instances we won't be asked to have the whole research routing procedure. We are able to check the anti finger table to research if all of us have a node in it whose value is greater than essential. Then we could use that node to retrieve the data connected with that key.

We can go through the ordinary research procedure as we did inside the original CHORD algorithm, if an unique key wasn't found because nodes anti finger table entry. But as we will likewise be looking in the anti-clockwise direction as we do in clock-wise direction, we shall get the important earlier than earlier algorithm as we're searching it in both the ways.



Start	Int.	Succ.
1	[1, 2)	1
2	[2, 4)	3
4	[4, 0)	6

Start	Int.	Succ.
2	[2, 3)	3
3	[3, 5)	3
5	[5, 1)	6

Start	Int.	Succ.
7	[7,0)	0
0	[0,2)	0
2	[2,6)	3

Start	Int.	Succ.
4	[4,5)	6
5	[5,7)	6
7	[7,3)	0

Figure Anti Finger Table

IMPLEMENTATION

Both principal execution metrics we discuss here is the first clockwise Chord regimen based on Open Chord, and the adjustment of software to match the characteristic of bi-directional routine. Moreover, we add some attributes for each application to weight the functionality of each formula.

The Open Chord platform can be an open source platform for Chord formula simulation. We use My Eclipse 8.0 to build up the, the source code can be gathered to some.bat document, we could found the command-line to determine the effects of adjustment produced within the source code.

Since the basic edifice of the finger table remains the same as unique Chord, we do not need to adjust the declaration of finger table classification. The number of entries in the finger table is m at most, and the dimension of finger table is $O(\log N)$, where N is the total number of nodes in the network.

Also, each nodes preserve an anti-finger table, which has m entries stored at the majority, the size of anti-finger table is $O(\log N)$. In the BiChord lookup formula, when a node send an internet search message to look up the desired key, if there is no items within the hand table and antifinger table that's closer than it self, the node is hence the predecessor or heir of the important. Otherwise, criteria will assess the routing table (fingers and anti-fingers), then the lookup message will be sent to the next hop node that is closer than the current node. The lookup operation is iteratively executed until it discovers the node that's previous or succeeding the wanted key.

Because the size of simulator is little (normally less than 50nodes), and also the simulator is centered on Java Virtual Machine (JVM), the research speed during the Chord circle is comparatively large. If all of us decide on a timer to consider the functionality of lookup operation in every criteria, the greatest accuracy we can achieve is nanosecond. Nevertheless, after several attempts, we locate the timer isn't the best way to compute the price since it's incorrect. Therefore we facilitate the jump counter here to compute the steps of searching for. Because the hops will never be impacted by the hardware settings, the counter can explicitly reveals the effectiveness of the formula within this situation. The pseudo code for the bi-directional lookup algorithm can be as following:

```
        Node findPredecessor(key,n){
        Node pred=n.getPredecessor();
        if (pred==null)
                return n; //n is the current node
        else if (key.isInInterval(pred.ID, n.ID) //check if the
        key is between the pred and current node
                return n;
                else                                    {Node
        n'=getClosestPrecedingNode(key) //if not,
        track the closest preceding node and lookup again
                return findPredecessor(key,n') // recursively
        find the predecessor of node n'
                }
        }
        Node findSuccessor(key,n){
                Node succ=n.getSuccessor();
                if (succ==null)
                 return n; //n is the current node
        else if (key.isInInterval(n.ID, succ.ID) //check if the
key is between the current node and successor's node
        return n;
        else {
        Node n'=getClosestPrecedingNode(key) //if not,
track the closest preceding node and lookup again
        return findSuccessor(key,n') // recursively find
the predecessor of node n'
}
}
Set<Serializable> retrieve_R(key){
        hops_R=0; //initialized the hops counter in anti-
finger table direction
        whild(!retrieved){
                Node responsible Node_R=null;
                responsible Node_R = findPredecessor(id);
                hops_R+=1; //while not retrieve the desired
        key,
        add the hop counter by 1
                try{
        result_R = responsibleNode_R.retrieveEntries(id);
// get the responsibleNode to fetch the entry
                retrieved = true; //if successfully get the
        value,
        set retrieved state to true
                }catch(Exception e){}
                        continue;
                        }
                }
                if(result_R                              !=null)
                values1.add(entry.getValue());
//add the lookup result to the valueset
        final_hopsR=hops_R; //get the hop counter for the
current lookup operation
                return values1;
        }
        Set<Serializable> retrieve(key){
                hops=0; //initialized the hops counter in
        finger table
                direction
                        whild(!retrieved){
                        Node responsibleNode=null;
                        responsibleNode                 =
                findSuccessor(id);
                        hops+=1; //while not retrieve the
                desired key, add
the hop counter by 1
        try{
           result = responsibleNode.retrieveEntries(id);
        //get the responsibleNode to fetch the entry
```

```
                        retrieved = true; //if successfully get the
                value,
                set retrieved state to true
                        }catch(Exception e){}
                        continue;
                        }
                }
                if(result !=null) values.add(entry.getValue());
// add the lookup result to the valueset
                final_hops=hops; //get the hop counter for the
current lookup operation
                return values;
}
```

It really can be observed within the pseudo code that within our changed version of Chord, we place a jump counter for research procedure in inverse and clockwise direction - clockwise direction. When crucial recover operation is executed, a hop value will be returned by system for both seeking routine.

EVALUATION

In this section, we evaluate the performance of Bi-Directional Chord with OpenChord simulator. The algorithms used in this test are the modified Bi-Directional Chord algorithm and also original OpenChord Chord criteria. The Note algorithm can be used for comparison.

Experimental Test Results :

Routine Table Size

Amount of entries in antifinger table and hand table will help estimate the total size of routine table.

Within the test, we simulated a community in OpenChord with [1,2,4,8,16,32,48] nodes respectively. In each dimension of community, we report the amount of nodes, a fair number of entries in hand table, a fair variety of records of antifinger table, the lookup hops in every single direction as well as the total variety of entries in regimen table.

Not surprisingly, the number of items in table is lessor equal to 2m, where m is the amount of nodes in the community. As well as the size of the table is about the total items number of whole community.

Look Up Hops

We pick up a same hunting key (a1) to search in an identical node (node0) to execute the look up procedure in Chord and BiChord formula, to consider the performance of program table look up. We document the trips counter for each size of system tested. The less trips means the algorithm retrieves the node with type in shorter measures.

Their evaluation explanation as well as the check data stand is given as follows:

Firstly we have attempted to calculate the value of number of hops in the present note protocol, Therefore the stand for that is given in Table 2

Table 2. Evaluation table for chord protocol

Number of	Finger table	Finger table size	Hops
1	0	0	1
4	2	8	1
8	3	24	1
16	4	64	1
32	5	160	2
48	6	288	2

Now for the new bidirectional finger table evaluation is based on the table given below

Table 3. Evaluation of bidirectional finger table

Numbe	Finge	Finge	Revers	Revers	Hop	Hops(R	Total
1	0	0	0	0	1	1	0
4	1	4	2	8	1	1	3
8	2	16	3	24	1	1	5
16	4	64	3	48	1	1	7
32	5	160	5	160	2	1	10
48	6	288	5	240	2	2	11

As it is obvious from both tables that the recent protocol that is applied using the bi-directional finger table requires a lesser variety of hops, when we look in the number of nodes worth = 32. But it's impossible to virtually assess this using a less amount of nodes. To obtain the best outcomes we must examine this within an environment where the number of nodes are comparable to a sensible network.

CONCLUSION

This statement proposes bi-directional research algorithm predicated on the OpenChord simulation platform. Throughout designing the formula we assemble an anti finger table. In the experiment we understand that the lookup performance is slightly enhanced. In addition, we offered some investigation to reveal the advancement of the research efficiency. In the future function we need to enhance the formula to implement the use of automatically picking the search direction by maintaining a check in the lookup hops.

REFERENCES

[1] Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications Ion Stoicay, Robert Morrisz, David LibenNowellz,David R. Kargerz, M. Frans Kaashoekz, FrankDabekz

[2] Using bidirectional links to improve peer-to-peer lookup performance JIANG Jun-jie†1, TANG Fei-long1,PAN Feng1, WANG Wei-nong2 (1Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200030, China)

[3] Wikipedia : http://en.wikipedia.org

[4] http://compnetworking.about.com/od/p2ppeertopeer/.../p2pintroduction