Novel Search Schemes for Distributed Cooperative Data Centers

To cite this article: Liang Dong and Xiaolu Zhang 2017 J. Phys.: Conf. Ser. 910 012022

View the article online for updates and enhancements.
Novel Search Schemes for Distributed Cooperative Data Centers

Liang Dong\textsuperscript{1,a} and Xiaolu Zhang\textsuperscript{2,b}

\textsuperscript{1}Yunnan observatory, Chinese academy of science, Kunming, China
\textsuperscript{2}School of Information Science and Engineering, Yunnan University, Kunming, China
Email \textsuperscript{a}dongliang@ynao.ac.cn, \textsuperscript{b}zxl@ynu.edu.cn

Abstract. Distributed cooperative data centers provide a new data storage and data processing architecture over heterogeneous physical nodes. The major challenge faced by such architecture is to find the right data on some node efficiently. In this paper, we present two alternatives for data indexing based on Chord, a Peer to Peer (P2P) overlay structure, which redefine the routing table structure that underlies the Chord and achieve small search paths. First, we present a regional search algorithm that routes data keys queries by region and super-node information. The search process can be easily implemented via \(O(\log K)\) hops, while maintaining \(O(\log K)\) routing information with \(K\) regions in the Chord ring at each node. We further propose a Two-hop search based on the regional search scheme which aims to reduce the average search paths to a constant with \(O(\log K)\) routing states about super-nodes. Results from theoretical analysis and simulations show that our improved routing algorithms can achieve higher search efficiency and the improved membership maintenances can keep routing information sufficiently up-to-date to validate higher search successful rate.

1. Introduction

With rapid increasing of big data, distributed cooperative data centers [1] have been the hotpot of current researched. Heterogeneous and variable data sources from users, companies, sensors and objects are widely distributed among data centers. Owing to the great volume and distribution of data as well as concurrent access of massive users, data indexing in data centers presents many new challenges: (1) the great volume of index data requires large-scale distributed storage for index data. Therefore, an efficient search scheme is needed to accurately locate storage nodes and rapidly find out index data; (2) concurrent access of massive users does not allow performance bottleneck in the search process, so as to guarantee high efficiency and concurrency of the search operation. However, the traditional indexing methods are not efficient for the distributed cooperative data centers due to the high computing complexity. For designing a good data search scheme is necessary. P2P overlay mechanism is usually considered as a technology in which data or programs are stored across a large number of nodes in self-organizing way is more successful than centralized approaches. The basic advantage over other approaches is that each node has equal opportunity to provide services with equal capabilities and roles. Chord protocol [2] as one of the popular structured P2P overlays provides an efficient search speed with less routing information, and has received many attentions to its good scalability, robustness and high search accuracy in distributed system. Yin et al. [3] researched on DHT-based Chord to address the load balancing of distributed file system in cloud computing; Kim et al. [4] presented DChord search scheme which does not depend on the randomness assumption to achieve its performance and deploys it to the Ubiquitous Sensor Network; Ding et al. [5] proposed an efficient quad-tree QT-Chord structure for global index of cloud data management; Zou et al. [6] improved Chord to support a multi-tenant indexing mechanism in clouds.
In a $N$-node Chord system, each node need maintain $O(\log N)$ other nodes information and $O(\log 2N)$ messages to guarantee the correct routing as nodes join and leave the system, and the average search paths is $1/2 \ O(\log N)$. Maintaining small routing tables helps keep the amount of messages required to deal with membership change small. However, this design pays a high price for having low search efficiency since each search needs contacting several nodes in sequence which consumes lots of bandwidth to receive neighbor nodes information. And another deficiency is that the Chord depends on the some certain nodes’ routing information without considering the load ability and diversity of nodes.

Chord has been extensively studied over the last several years. In particular, only a small number of algorithms have been proposed to provide efficient routing when the system meets churning and the membership is changing rapidly. Gupta et al. [7-8] describes a one hop routing scheme that every node maintains a full routing table containing information about every other nodes in the overlay. This approach reduces search latency to $O(1)$ hop but increases the bandwidth requirements since the messages of membership change. When nodes join and leave, the scheme need disseminate information about membership changes quickly enough so that nodes maintain accurate routing tables with complete membership information. The Kelips system [9] resolves the data search with $O(1)$ hop which consists of $K$ virtual affinity groups. Each node uses $O(\sqrt{N})$ space to store other nodes information of the same affinity group and $K-1$ nodes from other $K-1$ foreign affinity groups. It uses gossip scheme to update routing information. The scheme reduces the search paths by increasing memory usage and complexity of communication overheads. Bidirectional routing scheme [10] focuses on constructing two routing tables in dual directions, clockwise and anti-clockwise. The average search paths can approach $1/3 \ O(\log N)$, but the space of routing table is twice than Chord’s.

Most obviously, these schemes rely on increasing space for storing other node information to reduce the search paths. In a highly dynamic system, to maintain correct routing process, every node must send information to other nodes to make sure they are still alive, and replace them if not. Above methods make tradeoffs among the amount of storage overhead at each node, the search hops, and the maintenance cost of member change. In this paper, we propose two improved algorithms based on Chord to achieve the high search performance with low maintenance overhead. The regional search algorithm is to design a faster search $O(\log K)$ with $O(\log K)$ routing information, where $K$ is the number of regions in the Chord ring. The Two-hop search is aimed to reduce the search paths to a constant hop, while not increasing routing table size. Routing information per node need be corrected in order for the two algorithms to guarantee correct routing of queries. Thus, the Membership maintenances are designed to maintain accurate routing information when nodes join and leave arbitrarily.

2. Overview of Chord
Chord [2] uses consistent hashing functions such as SHA-I to distribute data keys over uniform nodes responsible for them in a $2^n$ identifier ring. Every node stores information of its predecessor and successor on the identifier ring. Furthermore, it maintains a routing table called finger table with $m$ other nodes entries. The definition of the finger table is shown in Table I (adapted from [2]).

| Notation       | Definition                                      |
|----------------|-----------------------------------------------|
| finger[i].start| $(n+2^i \mod 2^m), 1 \leq i \leq m$           |
| .interval      | $[\text{finger}[i].\text{start}, \text{finger}[i+1].\text{start})$ |
| .node          | first node $\geq n.\text{finger}[i].\text{start}$ |
| successor      | the next node in the Chord ring; $\text{finger}[1].\text{node}$ |
| predecessor    | the previous node in the Chord ring            |

Chord performs search in $O(\log N)$ hops, where $N$ is the number of nodes in the system. A node forwards a query for a given key to the node in its finger table with the closest identifier less than $k$. An example is presented in Figure 1 (adapted from [1]), whereby node 3 performs a search for key 1. Node
3 invokes the find_successor operation for this key, by visiting node 0, which eventually returns the successor of that key, i.e., node 1.

In order to guarantee the correctness of search and low latency for location, the Chord protocol uses a stabilization scheme running periodically to update finger tables and successor nodes in a dynamic environment.

In Chord, the structure of finger table ensures that node can always forward the query at least half of the remaining identifier space distance to \( k \), which accelerates the search process. In fact, each node only maintains a small amount of other nodes, and knows little about far-distant nodes. In this case, Chord node can’t find key’s successor immediately, thus, routing query is transmitted among some intermediate nodes. On the other hand, nodes are distributed unevenly in the interval. The hash function balances the load, but the different sizes of intervals disrupt it. When variable \( i \) is small, none or very few nodes distribute in the interval. It will double as variable \( i \) increases. Note that the search process depends on the finger table; therefore, we introduce two improved search algorithms based on super-node and region information aim to achieve high search efficiency while not increasing the size of routing table.

![Figure 1 Finger tables and search process for key 1](image)

3. Regional search

This section presents the design of our novel algorithm. The Chord ring is divided into \( K \) equal-sized regions, and every region selects a super-node based on its capabilities (i.e., lower latency, and higher bandwidth etc.). We will inherit the predecessor and successor list of Chord and add region information to routing table which maintains only about \( O(\log K) \) other nodes, called the Rfinger table. The Rfinger table and super-node will help require only \( O(\log K) \) search paths. When nodes join and leave, the routing table must be in an up-to-date state, so we outline a membership maintenance mechanism to ensure the correctness of search.

3.1 Reconstructing routing table

We divide a \( 2^m \) identifier ring into \( 2^d \) equal regions, let \( d \) be the number of bits of region identifiers. Assuming \( S_i \) is the super-node of the \( i^{th} \) region, \( S_i \) is responsible for maintaining all nodes information in the \( i^{th} \) region. The following definitions are applied in reconstructing routing table.

**Definition 1:** \( r_i \) is the region pointed by the \( i^{th} \) entry in \( n \)'s Rfinger table.

\[
r_i = (k + 2^{-i}) \mod 2^r, \quad 1 \leq i \leq d
\]

where \( k \) is the region of \( n \) belonging to.

**Definition 2:** \( g[i] \) is the space of region \( r_i \) in the ring.
\[ g[i] = \left( r_i \times 2^n, (r_i + 1) \times \frac{2^n}{2^m} \right), \quad 0 \leq r_i \leq 2d \]

**Definition 3:** \( P_i \) is the next hop node pointed by the \( i^{th} \) entry in \( n \)'s Rfinger table,

\[ P_i = (n + 2^m \times \frac{2^n}{2^m}) \mod 2^n, \quad 1 \leq i \leq d, \]

Let \( G_i \) be the set of all nodes in \( g[i] \) region, and \( S_i \) be the super-node of \( g[i] \). If \( P_i \in G_i, n.Rfinger[i].node=P_i \), or \( P_i \notin G_i, n.Rfinger[i].node=S_i \).

The Rfinger table of node 1 is shown in Figure 2, which node 1 belongs to region 0. Another case shown in Figure 3, node 33 is not in region 4, so we choose the super-node S35 of region 4 as Rfinger[3].node.

**Figure 2** Rfinger tables and search process without Super-node

**Figure 3** Rfinger tables and search process with Super-node

Apart from the Rfinger table, each super-node also maintains a super table including full super-nodes information of other regions mainly including variable region, ID identifier and IP address, and a region table containing other ordinary nodes of its region including variable ID identifier and IP address. Examples are shown in Table II and III.
Table 2 The super table of S60

| Region | ID  | IP   |
|--------|-----|------|
| 0      | S2  | IP2  |
| 1      | S10 | IP10 |
| 2      | S17 | IP17 |
| 3      | S30 | IP30 |
| 4      | S35 | IP35 |
| 5      | S41 | IP41 |
| 6      | S50 | IP50 |
| 7      | S60 | IP60 |

Table 3 The region table of S60

| ID | IP   |
|----|------|
| 57 | IP57 |
| 62 | IP62 |

3.2 Routing algorithm

We can find that the Rfinger table increases the distance of search paths while not reducing the routing information. Since every super-node contains a few nodes routing information to perform fast routing. The pseudo-code of search process is described as a function \( n.\text{find}\_\text{nexthop}(id) \) in Figure 4, where \( n \) is the node to invoke a search query, \( id \) is the target identifier. The parameters we use in our algorithm as follows:

- \( 2^m \) is the Chord identifier space.
- \( 2^d \) is the number of regions.
- \( k \) is the region which node \( n \) belongs to.
- \( S_k \) is the super-node of region \( k \).
- \( r \) is the region which \( id \) belongs to.
- \( S_r \) is the super-node of region \( r \).
- \( 2^m \) is the Chord identifier space.

Table 4 The super table of S35

| Region | ID  | IP   |
|--------|-----|------|
| 0      | S2  | IP2  |
| 1      | S10 | IP10 |
| 2      | S17 | IP17 |
| 3      | S30 | IP30 |
| 4      | S35 | IP35 |
| 5      | S41 | IP41 |
| 6      | S50 | IP50 |
| 7      | S60 | IP60 |

Figure 2 illustrates an example of search process on an identifier ring with \( m=6 \), \( d=3 \). Suppose node 1 wants to locate the identifier 62. It first checks whether the target and itself are in the same region, if not, it will send a request to node 33 whose identifier is closest to the target in its Rfinger table. Similarly, the request is transmitted in the Chord ring until it arrives at node 57 which is in the same region 7 with the target. Then node 57 sends a request to super-node S60. In the end, S60 finds the target in its region table. Figure 3 shows another search condition with the help of super-node. Node 1 sends a request to super-node S35 through its Rfinger table. S35 searches its super table (Table IV) for
the super-node S60 of region 7. S60 will check its region table to return the target. This faster search process will be presented as a further improved algorithm in section 5.

```plaintext
n.find_nextHop(id);  // if node n is the target id
if n=id then
    return n
end if  // if node n is the super-node of region r
if n=S_r then
    find (id ) in n’s region table
    return id
end if
else  // if id and node n are in the same region r, but node n is not the super-node of region r,
if n≠S_r and k=r then  // forward query(id) to S_r
    forward ( query (id), S_r)
    return S_r
end if
else  // if node n is the super-node of region k, but id and node n are not in the same region
if n=S_k and k≠r then  // forward query(id) to n.
supertable[k].id
end if
else  // if above conditions are not satisfied
    for j=d downto j=1 do
        if Rfinger[j].node is closest to id then
            forward ( query (id), n.Rfinger[j].node)
            return n.Rfinger[j].node
        end if
    end for
end if
```

3.3 Membership maintenance
When nodes join or leave, we must address the question that change information should be disseminated to all nodes in the Chord ring quickly so that nodes can maintain accurate routing tables to ensure the correctness and efficiency of search.

We imitate the process of One-hop [5] to build our membership maintenance mechanism. Every node sends keep-alive messages to its successor and predecessor nodes periodically. If either successor or predecessor does not respond, it decides to send a membership change message to the super-node of its region to notify the invalid node. After $t_{\text{change}}$ time, the super-node sends the change information about invalid nodes which are collected in last $t_{\text{change}}$ time to all other super-nodes. Change messages received by a super-node and a notify message are bound and forwarded to all nodes of its region after $t_{\text{wait}}$ time. In this hierarchical way, every node in the ring receives the change information within $t_{\text{change}} + t_{\text{wait}}$ time.
3.4 Bandwidth analysis

This improved algorithm performance depends on the number of regions \( K \). Consider above membership maintenance, we analyze the bandwidth use is mostly consumed on the change message data both super-nodes and ordinary nodes. In our system, the upstream bandwidth of super-node is the dominating and limiting factor. We will compute the value for \( K \) in a way that reduces the bandwidth utilization of super-node. The \( m \) bytes will be described the change event, and the overhead per message (or the size of acknowledged message) will be \( p \). \( r \) is the excepted rate of membership changes in this system. The excepted number of events per second in a region is \( r/K \). When a node detects a change event, it sends a notification to its region super-node. Therefore, the downstream bandwidth utilization of super-node is

\[
\frac{r \times (m + p)}{K}.
\]

Since each message must be acknowledged, the upstream utilization has

\[
\frac{r \times p}{K}.
\]

In the region, the size of change message sent from one super-node to another batches together events that occurred in the last \( t_{\text{change}} \) seconds is

\[
\frac{r}{K} \times t_{\text{change}} \times m + p.
\]

During \( t_{\text{change}} \) period, a super-node sends this message to all other super-nodes \((K-1)\), and receives an acknowledgement from each of them. Therefore, the upstream bandwidth comes out to

\[
\left(\frac{r \times m}{K} + \frac{2 \times p}{t_{\text{change}}} \right) \times (K - 1).
\]

After \( t_{\text{wait}} \), a super-node forwards a \textit{notify} message to all ordinary nodes, the aggregate message size becomes

\[
\left(\frac{r \times m}{t_{\text{wait}}} + \frac{p}{K}\right) \times \frac{n}{K}.
\]

The total upstream bandwidth on super-node is

\[
\frac{r}{K} \times (m \times n + p) + r \times m + \frac{2 \times p \times K}{t_{\text{change}}} + \frac{p \times n}{t_{\text{wait}} \times K}.
\]

By analyzing this function, we deduce the minimum where \( K \) is

\[
K = \sqrt{\frac{r \times m \times n}{2p} + \frac{n}{2t_{\text{wait}}} + \frac{r}{2} \times t_{\text{change}}}.
\]

This formula allows us to calculate the reasonable number of regions.

4. Algorithm analysis

Chord protocol is a binary search process via \( O(\log N) \) hops. Since the distance will halve by finding the closest node to the key from every node’s finger table iteratively. We use region information and super-node to locate the target rather than route by node’s successor. Our analysis follows a theorem.

\textbf{Theorem 1.} The average search paths of the regional search algorithm based on super-node is

\[
\text{Laverage} = \frac{1}{2} \log K + 2
\]
where $K$ is the regions in the ring.

Proof: The regional search algorithm based on super-node uses region information to locate key. The number of forwarding queries between regions distributes in the interval $[1, \log K]$. The search probability of every region is equal. Therefore, the expectation of the average search paths follows:

$$L_{\text{average}} = \sum_{i=1}^{1} P_i C_i = \frac{1}{K} \sum_{i=1}^{\log K} \left( \log K \right).$$

When a routing node finds itself in the same region with the target identifier, it sends a request to the super-node of its region. Given searching local information in region table, we can deduce the average search paths for the following value:

$$L_{\text{average}} = \frac{1}{2} \log K + 2.$$

5. Two-hop search

The improved regional search scheme presents a faster search $O(\log K)$ with maintaining an Rfinger table at each node. In this section, we propose a Two-hop search scheme. This scheme routes in two hops on average with $O(\log K)$ routing states on each node whose entry points to the super-node of region $k$.

5.1 System design

Our design is based on a structure like that used in improved regional search. The ring is split into $K$ equal-sized regions and each region is assigned to one super-node. Every node is required to store $O(\log K)$ routing information. The next hop pointed by the $i^{th}$ entry is redefined to super-node $S_i$ belonging to $i^{th}$ region, $n.Rfinger[i].node = S_i$, $1 \leq i \leq d$. Every super-node also keeps routing information for all other super-nodes. Figure 5 gives an example to show the Rfinger tables of node 1 and super-node 35.

5.2 Query routing

When a node wants to query the target key, it first checks whether the key and itself are in the same region, then sends this request directly to the super-node of the region. Otherwise, it finds the Rfinger table and forwards the request to next hop super-node whose region table contains the key. In the worst
case, it chooses the super-node whose identifier is closest to the target in the Rfinger table. In response, the chosen super-node examines its super table and forwards the request to the super-node responsible for the key. For example, in Figure 5, node 1 requests the target identifier 38 (belongs to region 4). It searches its Rfinger table to identify the super-node S35 which belongs to region 4, then, forwards the request to S35, who in turns looks up its region table to find the target.

5.3 Membership maintenance

The mechanism addressing the membership change (i.e., when node joins or leaves) is similar to the improved regional search. The difference occurs in what every node routing table only contains super-nodes information. We optimize the Two-hop membership maintenance for ordinary node and super-node, respectively. The goal is to propagate change information fast and reduce bandwidth consumption. Every node sends keep-alive message to its successor and predecessor nodes periodically. If an ordinary node fails, the scheme only sends change message to the super-node of its region and update the region table. In our scheme, every super-node also sends keep-alive message to all other super-nodes periodically. If any super-node does not respond, it sends a change message to all other super-nodes. After receiving the message, a super-node will forward it with an attached notify information to all nodes in its region.

6. Simulation results

In this section, we simulate the regional search and Two-hop algorithms on P2PSim [21]. P2PSim is a free, multi-threaded and discrete event simulator which supports several P2P protocols, for instance, Chord, One-hop, Kelips, etc. We reconstruct the routing table and modify routing algorithm on Chord protocol to evaluate our improved algorithms on P2PSim with specific topologies, and make a comparison with Chord. We simulate the experiments with 128, 256, 512, 1024 nodes generated by E2Egraph Kingdatas with 8, 16, 32, 64 regions, respectively. Nodes are uniformly distributed in a $2^{16}$ identifier space we preset. In the simulation, we use the middle node of a region as a special super-node. When a super-node fails, its successor which has a copy of its routing information soon detects and becomes new super-node. During the simulation, requests are generated by nodes in the P2PSim for randomly chosen keys with some nodes joining or leaving the overlay at random time.

Figure 5 shows the average search paths. We can see that the two improved algorithms significantly shorten the search paths than chord. And the average search paths in Two-hop are correctly completed by two or three hops. We also simulate the experiment with 1024 nodes to determine the failure rate. The result is shown in Figure 7. In the first 500 seconds, most of new nodes join rapidly. After approximately the first 1000 seconds, the query failure rate stayed steadily. Apparently, the failure rate of Two-hop search is lower than the regional search. This happens because Two-hop search only fails on routing table entries whose information is lost in super-node failure. In our simulation, we further examine the bandwidth consumption of super-node during query and membership maintenance. Figure
8 indicates that the bandwidth use of Two-hop search is lower due to the optimized membership maintenance for two kinds of nodes (ordinary node and super-node).

![Figure 7](image)

**Figure 7** The bandwidth utilization of per super-node

### 7. Conclusion

This paper presents two novel search algorithms based on Chord for the distributed cooperative data centers. We argue that existing improved algorithms are inefficient and inadequate for reducing search hops by increasing routing table size and maintaining more nodes information. In contrast to existing methods, we design our algorithms based on region information and super-node. In a N-node overlay network and K-region of Chord ring, each node maintains $O(\log K)$ other nodes information, and each super-node is responsible for maintaining all other super-nodes routing states. We also present two improved maintenances to disseminate information about membership changes quickly enough so that nodes maintain correct routing tables. Our theoretical analysis and simulation results confirm that these improvements can achieve lower search failure rate and provide more efficient search than Chord, which route $O(\log K)$ hops in the regional search algorithm and a constant search in Two-hop, respectively. We believe that the key contribution of this paper is to serve as a reference of the distributed cooperative data centers in which the Chord topological structure and the improved search algorithms can benefit the indexing mechanism.

### 8. Acknowledgment

This work is sponsored by Youth Fund of National Natural Science Foundation of China (11303094), National Natural Science Foundation Astronomy Joint Fund of China (U1431113), Applied Basic Research Program and Project of Yunnan Province of China (2015FB189), Western Light A Class Talent Program of China, and the Scientific Research Foundation of the Educational Department of Yunnan Province (No. 2015J0007).

### 9. References

[1] R. Basmadjian, P. Bouvry, G.D. Costa, et al, “Green data centers,” Large-scale Distributed Systems and Energy Efficiency: A Holistic View. John Wiley & Sons. P. 2015, pp. 159-196.

[2] I. Stoica, R. Morris, D. Karger, et al, “Chord: A Scalable Peer-to-Peer Search Service for Internet Applications,” ACM SIGCOMM Computer Communication Review, vol. 31(4), pp. 149-160, 2001.

[3] X. Yin, J. Yang, C. Qu, “Research on Load Balancing of Distributed File System in Cloud Computing,” Computer Science, vol. 41(3), pp. 141-144, 2014.

[4] C.S. Kim, S. Lee, J.I. Han, et al, “DChord: An Efficient and Robust Peer to Peer Search System,” Malaysian Journal of Computer Science, vol. 23(1), pp. 37, 2010.

[5] L.L. Ding, B.Y. Qiao, G.R. Wang, et al, “An efficient quad-tree based index structure for cloud data management,” In Proceedings of the 12th Conference on Web-Age Information Management, Berlin: Springer-Verlag, pp. 238-250, 2010.
[6] L.D. Zou, Q.Z. Li, L.J. Kong, et al, “Indexing Mechanism of Multi-Tenant Database Based on Chord,” Chinese journal of Computers, vol. 38(56), 2015.

[7] A. Gupta, B. Liskov, R. Rodrigues, “One Hop Search for Peer–Peer Overlays,” In Proceedings of the 9th Workshop on Hot Topics in Operating Systems, pp. 7-12, 2003.

[8] A. Gupta, B. Liskov, R. Rodrique, “Efficient Routing for Peer-to-Peer Overlays,” In Proceedings of the 1st Symposium on Networked Systems Design and Implementation, vol. 4, pp. 9-9, 2004.

[9] I. Gupta, K. Birman, P. Linga, et al, “Kelips: Building An Efficient and Stable P2P DHT through Increased Memory and Background Overhead,” Peer-to-Peer Systems II, Springer Berlin Heidelberg, pp. 160-169, 2003.

[10] P. Ganesan, G.S. Manku, “Optimal Routing in Chord,” In Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, pp. 175-185, 2004.

[11] G. Cordasco, A. Sala, “2-Chord Halved,” In Proceedings of the 2nd International Workshop on Hot Topics in Peer-to-Peer Systems, San Diego: IEEE Computer Press, pp. 72-79, 2005.

[12] D. Sun, Y. Wang, “Improvement of Chord routing algorithm,” Mechanical & Electrical Engineering Magazine, vol. 12, pp. 30, 2007.

[13] B.Q. Wang, P. He, “H-chord: Routing Model and Algorithm Implementation for Chord Based on Hierarchical Dividing,” Computer Engineering and Applications, vol. 42(36), pp. 141-143, 2007.

[14] G. Cordasco, L. Gargano, M. Hammar, et al, “F-Chord: Improved Uniform Routing on Chord,” Networks (Wiley InterScience), vol. 52(4), pp. 325-332, 2008.

[15] Y.J. Joung, J.C Wang, “Chord2: A Two-layer Chord for Reducing Maintenance Overhead via Heterogeneity,” Computer Networks, vol. 51(3), pp. 712-731, 2007.

[16] E.J.L. Lu, Y.F. Huang, S.C. Lu, “ML-Chord: A Multi-layered P2P Resource Sharing Model,” Journal of Network and Computer Applications, vol. 32 (3), pp. 578-588, 2009.

[17] X. Zheng, V. Oleshchuk, “Improving Chord search protocol for P2PSIP-based communication systems,” In Proceedings of the 9th International Conference on New Trends in Information and Service Science, pp. 1309-1314, 2009.

[18] R. Cuevas, M. Uruena, A. Banchs, “Routing Fairness in Chord: Analysis and Enhancement,” In Proceedings of the IEEE INFOCOM , pp. 1449-1457, 2009.

[19] G. Chiola, M. Ribaudo, “Neighbor-of-neighbor Routing over Deterministically Modulated Chord-like DHTs,” In Proceedings of the International Parallel & Distributed processing symposium, pp. 1-8, 2008.

[20] O. Babaoglu, M. Marzolla, M. Tamburini, “Design and Implementation of a P2P Cloud System,” In Proceedings of the 27th Annual ACM Symposium on Applied Computing, pp. 412-417, 2012.

[21] P2PSim: A simulator for peer-to-peer (P2P) protocols[EB/OL]. Available at : http://pdos.csail.mit.edu/p2psim/.