International Journal of Computational Intelligence and Informatics, Vol. 2: No. 4, January - March 2013

# RVN-Chord: A Novel P2P Lookup Algorithm for Dynamic Grid

**C Jeyabharathi**
*Research Scholar*
*Manonmaniam Sundaranar University*
*Tirunelveli*
*bharathi_guhan@yahoo.com*

**A Pethalakshmi**
*Head and Associate Professor of Computer Science*
*M.V.M Government Arts College (W)*
*Dindigul*
*pethalakshmi@yahoo.com*

***Abstract*- Grid computing is the next generation distributed system. Grid is an environment that allows sharing of resources that are managed by diverse, independent administrative organizations that are geographically distributed. An efficient resource discovery mechanism is one of the fundamental requirements for grid computing systems, as it aids in resource management and scheduling of applications. Among various discovery mechanisms, Peer-to-Peer (P2P) technology witnessed rapid development and the key component for this success is efficient lookup applications of P2P. Chord is a P2P structural model widely used as a routing protocol to find resources in grid environment. Since improvement on Finger table plays a vital role in efficient Grid Resource Discovery, our work contributes to reconstruction of finger table with new updates. In this paper, we proposed RVN-Chord (Recently Visited Node) which modifies the original Chord's finger table by adding a new column which stores the ID of Recently visited node. Every new lookup uses that ID to find the successor of the key and the search is minimal if the key matches with RVN.ID. We use NSC_SE simulator and Wireshark packet analyzer in our simulation. Theoretical and experimental analysis shows that RVN-Chord can reduce the number of hops per peer, message per peer, average communication time and memory consumed.**

Keywords- Grid Resource Discovery, P2P, Chord Protocol, RVN-Chord

## I. INTRODUCTION

As grid technique is growing rapidly in recent years, large-scale grid systems appear to provide the capability of flexible, secure, coordinated resource sharing, and problem solving among dynamic virtual organizations. These systems consequently are required to manage a large amount of related resources. In particular, the shared grid resources can vary from plain desktop systems to clusters and from storage devices to large datasets, even grid service could be seen as an extension of grid resource. Therefore, Grid Resource Discovery (GRD) plays a crucial role in the whole system [1], and its discovery Models and strategies have a vital influence on the grid system performance.

GRD can be classified into attribution matching and semantic matching pattern. In attribution matching pattern, GRD can also be classified into Centralized, Hierarchical, Peer-to-Peer based, and group-clustering models from the system architecture aspect. Among the models given above, centralized and hierarchical models are easier to design and manage, whereas they prove to be inefficient as the scale of grid system rapidly increase due to the bottleneck of servers and single point of failure. Therefore, the P2P-based model which combines the P2P technology into grid is applied widely to overcome those problems.

Grid and P2P [7] are the two most common types of resource sharing systems currently in wide use. These two systems are evolved from different communities and serve different needs. Grid systems interconnect computer clusters, storage systems, instruments, and in general available infrastructure of large scientific computing centers in order to make possible the sharing of existing resources, such as CPU time, storage, equipment, data and software applications. Most Grid systems are of moderate-size, are centrally or hierarchically administered and there are strict rules governing the availability of the participating resources. In

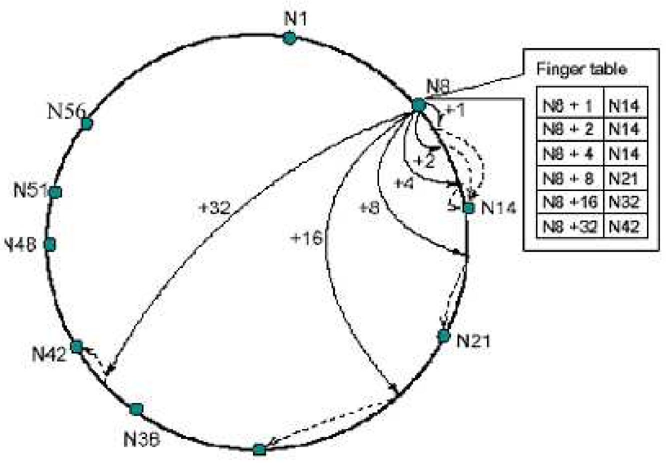


**Figure 1: Chord Ring with 64 node**

P2P system most resource location queries are not attribute-dependent as in Grids but they either specify a file name, i.e., keyword searches or range queries.

P2P network is a kind of distributed network, the network's participants to share their own part of the hardware resources (processing power, storage capacity, network connectivity, printer, etc.), which shares the resources required by the network services and content, can be directly accessed to by other peer nodes without intermediate entities. P2p network can be divided into centralized P2P networks, unstructured P2P networks and structured P2P networks. The structured P2P networks based on a DHT, which has the advantage of distributed and avoided the drawback of unstructured P2P system. It uses a structured hash algorithm, stored keywords to different nodes in the network by certain rules, through the corresponding routing algorithm to get the keyword. In a Distributed Hash Table, the more well-known agreement is Chord, Pastry, Tapestry, CAN and so on. Chord algorithm is relatively simple and easy to achieve. Among structured P2P lookup protocols, Chord has been researched and applied widely as a result of its simplicity, high efficiency and credibility.

This paper proposes an improved chord model which is called RVN-Chord. It is based on the addition of recently visited node id in the finger table. The simulation results show that the routing efficiency is highly improved by reducing number of hops, messages and communication time.

## II. Related Work

### A. *Chord*

Chord is proposed by Ion Stoica of the University of California and Robert Morris of lab of MIT et al., as a resource routing protocol based on DHT. The Chord protocol [6] supports just one operation: given a key, it maps the key onto a node. Depending on the application using Chord, that node might be responsible for storing a value associated with the key. Chord uses a variant of consistent hashing to assign keys to Chord nodes. In Chord, each node needs “routing” information about only a few other nodes. In the steady state, in an N-node system, each node maintains information only about O(log *N*) other nodes, and resolves all lookups via O (log *N*) messages to other nodes. Chord maintains its routing information as nodes join and leave the system; with high probability, each such event results in no more than $O(\log^2 N)$ messages. Three features that distinguish Chord from many other peer-to-peer lookup protocols are its simplicity, provable correctness, and provable performance. A Chord node requires information about O(log *N*) other nodes for efficient routing, but performance degrades gracefully when that information is out of date. This is important while practising because nodes will join and leave arbitrarily, and consistency of even O (log *N*) state may be hard to maintain. Only one piece of information per node need be correct in order to guarantee correct routing of queries.

### B. *Consistent Hashing*

The consistent hash function assigns each node and key an m bit identifier using a base hash function such as SHA-1. A node’s identifier is chosen by hashing the node’s IP address, while a key identifier is produced by hashing the key. The identifier length m must be large enough to make the probability of two nodes or keys hashing to the same identifier negligible. Consistent hashing assigns keys to nodes as follows. Identifiers are ordered in an identifier circle modulo $2^m$. Key *k* is assigned to the first node whose identifier is equal to or follows in the identifier space. This node is called the successor node of key *k*, denoted by successor (*k*). If identifiers are represented as a circle of numbers from 0 to $2^m$ -1, then successor(*k*) is the first node clockwise from *k*. Figure 1 shows an identifier circle with *m*=6 The circle has ten nodes: 1,8,14,21,32,38,42,48,51 and 56. The successor of identifier 1 is node 1, so key 1 would be located at node 1. Similarly, key 5 would be located at node 8, and key 53 at node 56.Consistent hashing is designed to let nodes enter and leave the network with minimal disruption. To maintain the consistent hashing mapping when a node n joins the network, certain keys previously assigned to n's successor now are assigned to n. When node n leaves the network, all of its assigned keys are reassigned to n's successor. In the example given above, if a node were to join with identifier 54, it would capture the key with identifier 53 from the node with identifier 56.

The Finger Table of Chord ring can be defined as an array of log(*N*) elements (where *N* is the total number of nodes in the overlay). It contains the addresses of some of the successors of the node along with the ring at distances that increase as a function of the index in the table. Chord requires each node to keep a "finger table" containing up to m entries. The *i*th entry of node *n* will contain the address of successor $((n+2^{i-1}) \bmod 2^m)$. With such a finger table, the number of nodes that must be contacted to find a successor in an *N*-node network is O (log *N*). The finger table is constructed as follows:

Table 1: Finger Table of Chord

| | |
|---|---|
| Finger[*k*].start | $(n+2^{k-1}) \bmod 2^m$ |
| Finger[*k*].interval | [Finger[*k*].start, Finger [*k*+1].start) |
| Finger[*k*].node | successor (Finger[*k*].start) |

## C. *Existing Variations of Chord*

Till recently, researchers have proposed many improved schemes to enhance routing efficiency of Chord Protocol. To deal with the routing delay and ignorance of physical topology information in P2P, [8] presents a topology-aware structured P2P system, named TB-Chord, which applies a suit of mechanisms to extend Chord to optimize the utilization of physical network topology and overlay network. The TB-Chord makes effective use of network topology structure during network routing. The result of experiments show that the TB-Chord, compared with traditional Chord, has obvious improvements in the routing delay and the hops of overlay networks. At the same time, TB-Chord does not need super node or plus layer which will pay more maintenance costs. Fan chao et al., [4] proposed BNN-Chord algorithm based on neighbors' neighbor (NN) chord. NN-Chord algorithm extends finger table using neighbors' neighbour link which is based as learn table. This table maintains information of the successor's successor node, which can reasonably increase finger density of routing table to find the neighbor node, which is more close to the object. The BNN-Chord algorithm effectively reduces the search path length and can improve the system performance.

*Chunling Cheng et al [2]* proposed advanced chord routing algorithm based on redundant information replaced objective resource table. The redundant information in the finger table of Chord ring not only takes valuable space, but also increases search delay. There is a direct idea that using inspired information to replace the redundant information can be used, so that each node can use the space effectively and establish links with much more nodes, and the idea can improve the search performance greatly. Towards an Efficient P2P system capable of processing multi-attribute Multi-keyword fuzzy-matching queries with High recall ratio and load balancing, *ZHAO Xiu-Mei et al.,* [10] proposed a new resource indexing model which is expanded from Chord and called MF-Chord. *Yufeng Wang et al.,* [9] analyzed the of Chord algorithm to reconstruct the finger table in Chord, in which counterclockwise finger table is added to achieve resource queries in both directions, and the density of neighboring fingers is increased. Additionally, AB-Chord implements a new operation to remove the redundant fingers introduced by adding fingers in AB-Chord. In comparison with original Chord (and Bi-Chord), new fingers are inserted into the middle of two neighboring fingers in clockwise and counter clockwise direction. *Huayun Yan et al.,* [5] modified the finger table of original Chord, and modified all the places which relate to the entry of the finger table, such as the node join, routing procedure etc. Huayan Yan et al., get a very perfect routing result through experiment and found that the routing hops is close to a constant number. The cost is more updating when joining a node and deleting a node in the system, and nearly double the size of finger table. This system will be more effective in a comparatively steady condition.

## III. Design of RVN-Chord

In this section, we proposed to use the modified chord protocol by adding an additional entry into the Base Chord finger table to store the location of Recently Visited Node (RVN). The next lookup will use this id to locate its key, if the match is found. The primary aim of any protocol is the efficient and fast lookup of nodes containing keys. Generally the performance of Chord like algorithm can be analyzed in terms of three metrics: the size of the finger table of every node, the number of hops a request needs to travel in the worst case, and average number of hops. In the original Chord, when a node requests a key, it has to search its own finger table first and it may find the successor of the key in that table if the match is found. Otherwise, the node sends messages to other nodes whose node id is less than or equal to the searched key and it may need few hops to locate it. After locating the successor of the key, the result is returned to the node who started the search and the lookup process is successfully done. Our protocol modifies the finger table of the original Chord to add a new entry in the finger table which stores the recently visited node's id.

In RVN-Chord, we are using SHA1 hash function to map key and node id as in the original Chord. Our modified finger table stores the Recently Visited Node id which reserved the key of the previous lookup. This id will be stored in all the nodes of the Chord ring. This is identified by the variable *RVN.id* which is the first column entry of any finger table. This *RVN.id* is similar to the use of Recent Document list in our computer file system. By using the recent profile, the next search will be easy and we find the document's location quickly if the search matches with the recent document. Similarly in our proposed model, for every new search the node starts lookup by checking *RVN.id* of its finger table which was updated by the previous lookup. If the *RVN.id* matches the query, the process will succeed and it can directly locate the destination. If it is not exactly matched with the node, next it is checked if the key is greater than the *RVN.id*. If it is true then the recent route id is in its destination path. Then it can jump to that id and continue its search from that location. Otherwise, the normal Chord algorithm is invoked to find the successor of key. This type of lookup will reduce the number of hops, messages and communication time. Since Memory consumed by the finger table to store the *RVN.id* is very less in terms of bytes, it will not be a big issue.

### A. *RVN-Chord Algorithm*

The following are the steps followed in our proposed method.

1) Construction of finger table

a). Add a new entry as follows.

*RVN.id = last visited node.id*

b) *Finger[k].start* : $(n+2^{k-1}) \bmod 2^m$

*Finger[k].interval* : *[Finger[k].start, Finger [k+1].start)*

*Finger[k].node* : *successor (Finger[k].start)*

2) Lookup of key using *Recently_visited_node.id*

```
a.  Check if key = Recently_visited_node.id  then
            Successor (key) = Recently_visited_node.id
    Else
            Check if key > Recently_visited_node.id then
  Jump to Recently_visited_node.id then continue the search.
```

3) If step 2 fails, use the normal search as in base chord method.

4) Call step 1(a) to update the finger table for every successful search.

5) When the visited node leaves the ring or any failure occurs, Set *Recently_visited_node = successor(Recently_visited_node)* and update all finger tables.

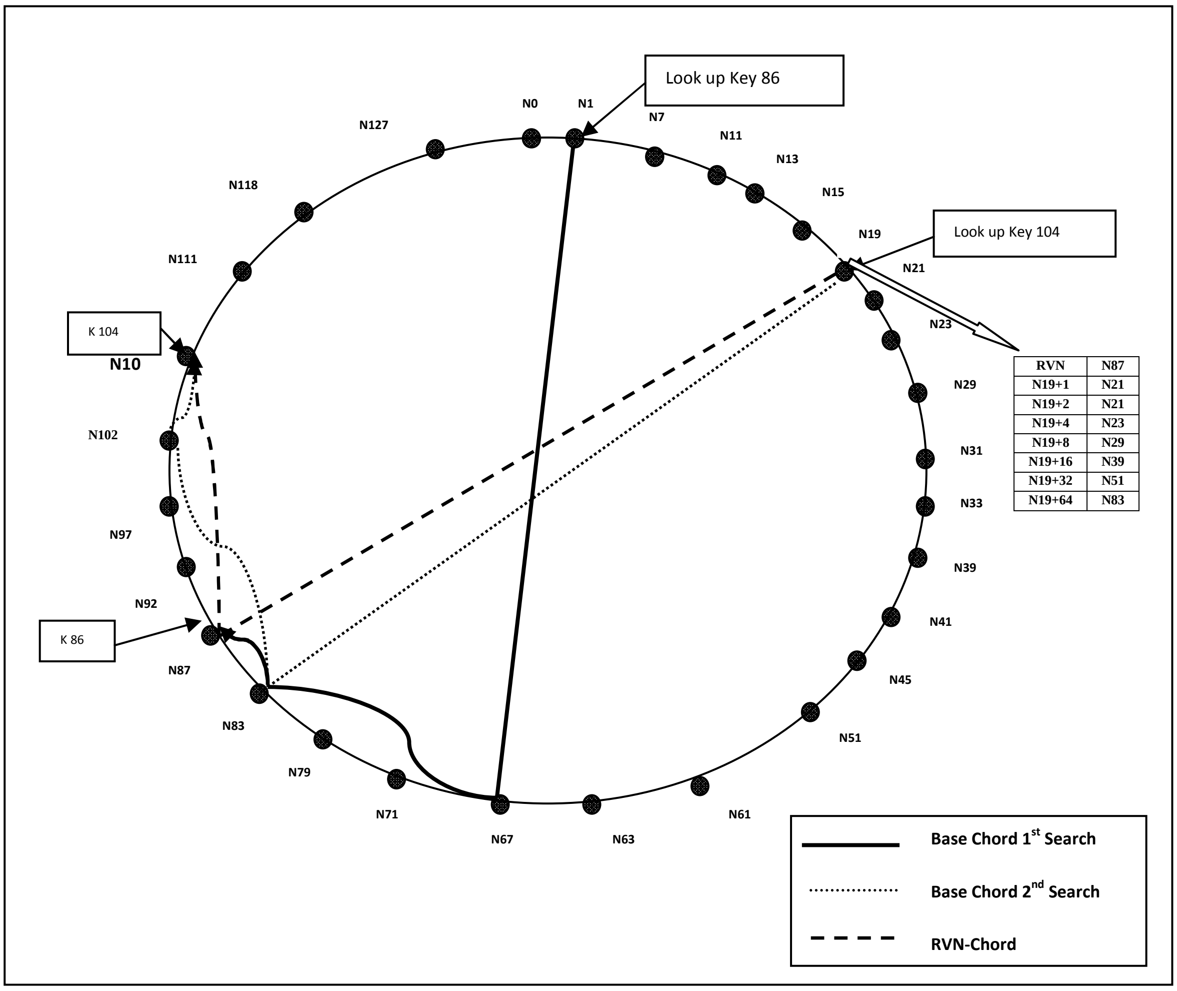
Next we present the Pseudo of RVN-Chord routing algorithm as follows:

```
// Ask node n to find the successor of id
n.RVN_find_successor(id)
      {
            if ( id == RVN.id )
               return RVN.id;
            else if ( id > RVN.id)
              start_from_RVN_find_successor(id);
            else
               n.find_successor(id);     // BaseChord Process
       }
start_from_RVN_find_successor(id)
          {
           for(i=RVN.id;i<=n.id-1;i++)
           if (finger[i] == id)
               successor(id)=finger[i];
           }
```

```
//Base Chord Process
n.find_successor (id)
if (id(n,successor])
   return  successor;
else
    n' = closest_preceding_node(id);
 n.closest_preceding_node(id)
    for i=m downto 1
        if(finger[i] (n,id))
            return finger[i];
     return n;
```

*Figure 2: Pseudo Code of RVN-Chord Algorithm*

## B. Example Analysis



| RVN | N87 |
|---|---|
| N19+1 | N21 |
| N19+2 | N21 |
| N19+4 | N23 |
| N19+8 | N29 |
| N19+16 | N39 |
| N19+32 | N51 |
| N19+64 | N83 |

*Figure 3: Chord Ring with 127 nodes*

Figure 3 depicts a chord ring whose m is equal to 7 (*m* is the bit size of the keywords or identifiers). The identifiers ranges from 0 to 127($2^7$-1). Here, we are comparing RVN Chord with the base Chord algorithm.

$$\begin{tikzcd} N1 \arrow[r] & N67 \arrow[r] & N83 \arrow[r] & N87 \end{tikzcd}$$

The number of hops and messages are compared between both models for searching a few keys. Consider node N1 searches key 86 in the Chord ring. In normal Chord, the process of N1 is looking up key K86 is

It needs *3 hops and 14 messages* to find the object node reserving the key N86. The next search can be started by any node and let us consider the node N19 starts the lookup of key 104. The process looks like

N19 → N83 → N102 → N106

and it needs *3 hops and 13 messages*.

Now we apply RVN-Chord to find the successor of the key. Assume that the node N19 look for the key 104. When the process is started RVN-Chord verifies its Recently Visited Node id for quick search, because it stored the last visited node in its finger table's first column. As per our previous example, the *RVN.id* is 87 (as in base chord). Node N19 first checks RVN.id for equality (if 104 = 87) and the match is not found. It then checks condition (N104 > N87) and it is true. So it jumps to N87. Next it searches the finger table of N87 and finds that N104 is the 5th finger of N87 which reserves the key 106. The lookup looks like

N19 → N87 → N106

This process requires only *2 hops and 6 messages* to locate the key whereas base Chord locates the key in *3 hops and 13 messages***.** The RVN-Chord algorithm performs well for the lookup of keys whose successor is equal to *RVN.id* or greater than *RVN.id*.

## IV. Simulation Analysis

We use the NSC_SE (Network Simulation Capture Special Edition) simulator to evaluate the performance of the proposed RVN-Chord. Motes, Androids, T-Engines, Nokia epocs, Hackman tool, UH_ABC (Universal Hack for All Block Ciphers), Linphone, Netsnooper, Wireshark and many more applications/libraries can be connected with NSC_SE. NSC_SE provides all information including connection establishing time, jitter, IP delay, Latency, Error rate, Pocket loss, Power Consumption and so on. Along with the Simulator we also use Wireshark, the free and open-source packet analyzer for network communication and to maintain all the required basic information of Chord protocol.

*Table 2: Average number of hops, messages and communication time in Chord, ML-Chord, RVN-Chord*

| **Messages** *:* | **Nodes** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **256** | **512** | **1024** | **2048** | **4096** | **8192** | **16384** | **32768** |
| **Chord** | 18.3 | 39 | 57.9 | 77.1 | 116.8 | 128.6 | 157.8 | 190.7 |
| **ML Chord** | 14.5 | 31.7 | 32.6 | 38.7 | 41.2 | 49.1 | 79.5 | 85.8 |
| **RVN Chord** | 10.6 | 27.2 | 29.1 | 35.5 | 36.7 | 43.9 | 67.3 | 75.9 |
| **Hops***:* | | | | | | | | |
| **Chord** | 4.4 | 6.3 | 6.8 | 6.3 | 7.8 | 8.2 | 9.1 | 8.9 |
| **ML Chord** | 4.3 | 5.5 | 6.7 | 6.2 | 7.3 | 7.2 | 7.3 | 7.5 |
| **RVN Chord** | 3.3 | 4.3 | 5.5 | 4.7 | 4.5 | 5.3 | 6.4 | 7.5 |
| **Communication Time***:* | | | | | | | | |
| **Chord** | 939.6 | 1072.2 | 1167.5 | 1226.4 | 1365.1 | 1473.2 | 1942.2 | 2409.2 |
| **ML Chord** | 928.4 | 964.5 | 1230.2 | 1283.6 | 1309 | 1325.1 | 1836 | 2563.2 |
| **RVN Chord** | 960.3 | 1072.8 | 1140.1 | 1228.8 | 1557.2 | 2052.3 | 767.9 | 834.6 |

We consider a chord network before and after improvement with N=256, 512, 1024, 2048, 4096, 8192, 16834, 32768. The comparisons among base Chord, ML-Chord [3] and RVN-Chord are made in terms of number of hops, number of messages per peer, average communication time and memory consumed (RAM

Bytes). As shown in table 2, the average number of messages, hops and communication cost of RVN-chord is better than ML-Chord and Chord.

## V. PERFORMANCE EVALUATION

Figure 4, 5, 6 and 7 shows the average hops, average communication cost, average messages and memory consumed for Chord, ML-Chord and RVN-Chord with changes of total number of nodes in P2P systems. Obviously, the average number of hops, communication time and messages are reduced in RVN-Chord. The memory consumed in terms of byte is the same as the existing Chord. The reason is that Recently-visited-node is used greatly for many lookups of the Chord to locate resources. We are updating the finger table for every successful search to maintain the Recently Visited Node meaningful.

From the simulation result, we can conclude that when compared to the Chord algorithm, RVN-Chord performs well in the case of using the previously visited node id or the same path.

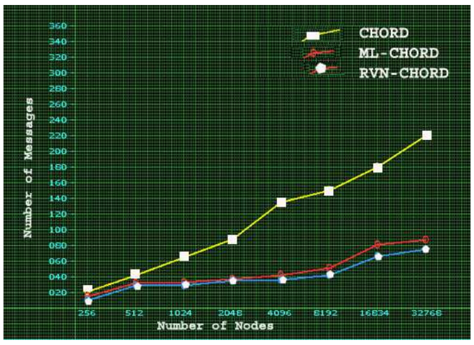


*Figure 4: Number of Messages per peer*

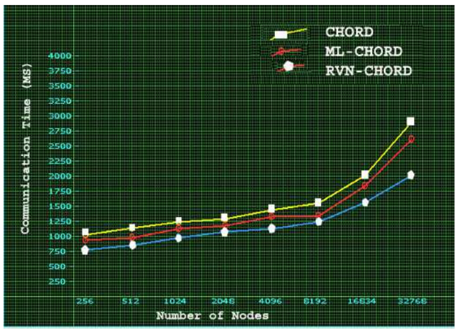


*Figure 5: Communication time*

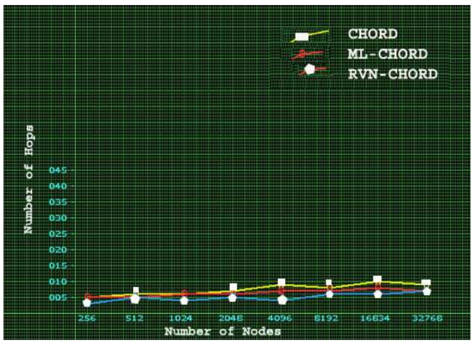


*Figure 6: Number of Hops per Peer*

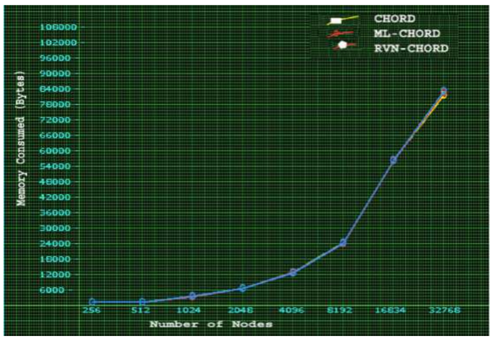


*Figure 7: Memory Consumed*

## VI. CONCLUSION AND FUTURE WORK

In recent years a lot of attention has been paid to the utilization of P2P technologies in Grid Resource Discovery. Since P2P approach will be suitable for dynamic distributed searching in a large-scale Grid environment, our research focuses on structured P2P protocols. In this paper we first studied Chord routing algorithm and a few proposed improvement strategies. We present the RVN algorithm based on the Recent – Route of the searched key. A new entry to represent the recent–route is added as the first column of the finger table. This algorithm will work efficiently if the new search needs the recent route. Our simulation result shows that the average number of hops is less than original Chord and ML chord. This algorithm performs well when the number of nodes is increased greatly. In our future research we have planned to incorporate Fuzzy rules into our RVN-Chord algorithm for efficient Grid Resource Discovery. We hope that the search performance of our modified RVN-Chord will be further improved.

## REFERENCES

[1] R.Buyya and S.Venugopal, "A gentle introduction to grid computing and technologies",CSI communications, vol.29, July 2005, pp.9-19.

[2] Chunling Cheng, Yu Xu, Xiaolong Xu, "Advanced Chord Routing Algorithm Based on Redundant Information Replaced and Objective Resource Table", in the proceedingsof Computer Science and Information Technology (ICCSIT), IEEE,2010.pp.247-250.

[3] Eric Jui-Lin Lu, Yung-Fa Huang, Shu-Chiu Lu, "ML-Chord: A multi-layered P2P resource sharing model", Journal of Network and Computer Applications 32, 2009,pp.578–588.

[4] Fan chao, Hongqi Zhang, Xuehui Du, Chuanfu Zhang, Zhengzhou,"Improvement of Structured P2P Routing Algorithm Based on NN-Chord", 7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), 2011. Pp. 1-5.

[5] Huayun Yan, Yunliang Jiang , Xinmin Zhou1, "A Bidirectional Chord System Based on Base-k Finger Table", International Symposium on Computer Science and Computational Technology,2008.pp.384-388.

[6] Ion Stoica, Robert Morris, I.Stoica, D.L.Nowell, D.R.Kargar, M.F.Kaashoek, and H.Balakrishnan"Chord: A Scalable Peer-to-Peer lookup service for internet applications", IEEE/ACM Transactions on Networking, vol 11, issue 1, pp 17-32, Feb 2003. (First appeared inProc. ACM SIGCOMM conference on Applications, Technologies, Architectures, and protocols for ComputerCommunication, pp. 149-160, 2001.

[7] P. Trunfio, D.Talia, C. Papadakis , P. Fragopoulou, M. Mordacchini , M. Pennanen, K. Popov, V.Vlassov, S. Haridi , "Peer-to-Peer Resource Discovery in Grids: Models and Systems", This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265). Preprint submitted to Elsevier Science,3 August 2006.

[8] Wei Lv1, Qing Liao2, Jingling Zhao3, Yonggang Xiao "TB_Chord: An Improved Routing Algorithm to Chord Based on Topology-aware and Bi-Dimensional Lookup Method", 978-1-4244-3693-4/09/$25.00 ©2009 IEEE.pp. 1-4.

[9] Yufeng Wang, Xiangming Li "AB-Chord: an efficient approach for resource location in structured P2P networks", 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on - Autonomic and Trusted Computing , 2012. pp.278-284.

[10] ZHAO Xiu-Mei, LIU Fang-Ai, Jinan "MF-Chord: Supporting Multi-Attribute Multi-keyword Fuzzy-Matching Queries", ITIME '09. IEEE International Symposium on IT in Medicine & Education, (Volume:1 )2009. pp. 522 - 527