Survey-based Comparison of Chord Overlay Networks

Alaa Arabiyat,
University of Jordan,
Amman, Jordan

Sulieman Bani-Ahmad
Al-Balqa Applied University
Salt, Jordan

Mohammad F. Ababneh
Al-Balqa Applied University
Salt, Jordan

ABSTRACT
Chord is a structured peer-to-peer (P2P) overlay network in which participating peers share resources as equals. To find a specific data item within the network, Chord system provide a lookup mechanism that matches a given key to a network node responsible for the value associated with that key. Chord is recently proposed to become one of the new approaches for building large-scale Internet applications. This paper aims to survey the Chord network, study its main characteristics, and compare its original performance with the performance of the enhancements being deployed over the original Chord.

General Terms
Distributed systems.

Keywords
Chord, Peer-to-peer, structured, lookup mechanism, Distributed hash table.

1. INTRODUCTION
This paper attempts to discuss the Chord structured p2p overlay network and compares the original Chord with other enhanced Chord networks in one characteristic or more.

An overlay network can be defined as a logical network on top of one or more networks. A popular example of such networks is the Internet. The main function of an overlay network is to provide means by which a large number of computing resources are connected together and accessed. And, as can be seen nowadays, various high-level distributed services can be built on top of an overlay network. The performance and efficiency of these high-level distributed services strongly depend on the characteristics of the underlying overlay network [1].

Peer-to-peer networks are overlay networks because their computing nodes operate on top of the Internet as in Figure 1. In structured P2P networks routing of messages is done using distributed hash tables (DHT) that is used to route messages to a node having a specific logical IP address, whose IP address is not known in advance [1].

In the simplest case, DHTs can be used to keep and save (key,value) pairs much like centralized hash tables. Lookup and join operations can be done in a small number of routing hops. The overlay network is self organizing, and each node keep only a small routing table with size constant or logarithmic in the number of existing nodes in the network [2, 3].

A hash-table interface is a good approach for a distributed lookup algorithm because it states few rules on the construction of keys or for data items and nodes. The main requirements are that data be identified using unique numeric IDs and those nodes are also identified using numeric IDs from the same space. This structure is different from that implemented in Napster and Gnutella (unstructured p2p overlay networks), which look for keywords, and assume that data is basically stored on the publisher's node. However, such systems could still make use of a distributed hash table—for example, Napster's centralized database recording the mapping between nodes and songs could be replaced by a distributed hash table [3].

A DHT supports just one function: lookup(key) results the ID (e.g., IP address) of the node that store currently the given key. A simple distributed storage application might use this interface as follows. Someone who wants to publish a file under a particular unique name would convert the name to a numeric key using an ordinary hash function such as SHA-1, then call lookup(key). The publisher would send the file to be stored at the resulting node. Someone wishing to read that file would obtain its name, convert it to a key, call lookup(key), and ask the resulting node for a copy of the file [3].A complete storage system would have to take care of replication, caching, authentication, and other issues; these are outside the immediate scope of the lookup problem.

Chord is considered one of the best candidates to design of peer-to-peer systems and applications because of its ability to address these difficult problems [4]:

Load balance: Chord uses a hash function that allocates keys to the nodes evenly; this results in natural load balance of data items on network nodes [4].

Decentralization: Chord is fully distributed; no node has higher capabilities or privileges than any other. This makes Chord appropriate for peer-to-peer applications [4].

Scalability: each node keep only a small routing table with size constant or logarithmic in the number of existing nodes in the network, so even very large systems can be implemented using Chord. [4].

Availability: Chord automatically renews its routing tables as a consequence to newly joined nodes as well as node failures, so the node keeping a key can always be found. [4].

Flexible naming: the Chord key-space is open. This gives applications the flexibility to map their own names to Chord keys according to some hashing functions. Neither limitations nor constraints are stated by the Chord system [4].

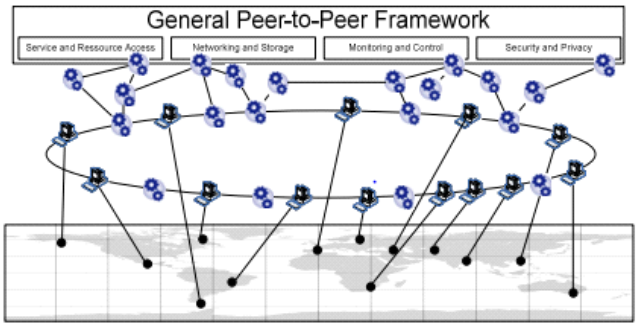

Figure 1. General P2P network [12].

2. The Base Chord Protocol

The Chord protocol state mechanisms on how to look up keys, how new nodes join the system, and how to recover from nodes failure or departure.

2.1 Consistent Hashing

Chord [4] uses consistent hashing function to map keys to its nodes. The consistent hash functions assign network nodes (peers) and data keys an *m*-bit identifier using SHA-1 as the base hash function. A peer's identifier is chosen by hashing the peer's IP address, while a key identifier is produced by hashing the data key. The length of the identifier *m* must be large enough to make the probability of keys hashing to the same identifier very small, they are usually selected to be 128 bit or 160 bit. Identifiers are ordered on an identifier circle modulo 2^m as a logical ring. Key *k* is assigned to the first peer whose identifier is equal to or follows *k* in the identifier space. This peer is called the successor peer of key *k*, denoted by *successor*(*k*). To maintain consistent hashing mapping when a peer *n* joins the network, certain keys whose IDs are equal or less than the ID of the joined node that were previously assigned to *n*'s successor now need to be reassigned to *n*. When peer *n* leaves the Chord ring, all of its assigned keys are reassigned to *n*'s successor and its become responsible for theses keys[4, 5].

2.2 Scalable Key Location

Each peer in the Chord ring needs to know how to locate its successor peer on the identifier circle. Lookup queries map data item key and NodeID. For a given identifier, it could be passed around the ring via the successor pointers until they find the peer with the desired identifier. It in turn returns its logical address and the requester could then contact it directly to get the desired data item. An example is illustrated in Figure 2, here peer 8 apply a lookup for key 54. Peer 8 run the find successor operation for this key, which returns the successor of that key, i.e. peer 56. The query passes every peer on the ring between peer 8 and peer 56. The response is returned along the reverse of the path. As *m* is the number of bits in the key/NodeID space, each peer *n* keep a routing table with up to *m* entries, called the finger table. The *i*th entry in this table at peer *n* contains the identity of the first peer *s* that succeeds *n* by at least $2^i - 1$ on the identifier circle, i.e., s = successor($n + 2^i - 1$), where $1 \leq i \leq m$. Peer *s* is the *i*th finger of peer *n*. A finger table entry includes both the Chord identifier and the IP address of the relevant peer. Figure 2 shows the finger table of peer 8, and the first finger entry for this peer points to peer 14, as the latter is the first peer that succeeds (8+20) mod 26 = 9. Similarly, the last finger of peer 8 points to peer 42, i.e., the first peer that succeeds (8 + 25) mod 26 = 40. In this way, peers store information about only a small number of other peers, and know more about peers closely following it on the identifier circle than other peers. Fingers drastically shorten the lookup path to O(logn) hops. Nodes periodically run a *fix fingers* () procedure to refresh the finger table entries. [4, 5].

2.3 Node Joins

When a peer joins the Chord system, the successor s and predecessors around the joined peer need to be updated. It is important that the successor pointers are up to date at any time because the correctness and efficiency of lookups is not accomplished otherwise. The Chord protocol uses a *stabilization* protocol [4] running periodically in the background to update the successor pointers and the pointers entries in the finger table. The correctness of the Chord protocol relies on the fact that each peer is aware of its successors [4, 5].

2.4 Node Failure

When peers fail, it is possible that a peer does not know its new successor, and that it has no chance to learn about it. To avoid this situation, peers maintain a successor list of size *r*, which contains the peer's first *r* successors. When the successor peer does not respond, the peer simply contacts the next peer on its successor list. Also, this procedure make it easy for key replication, instead of storing key k at only successor(k), it is replicated on the *r* successors of k. This way, even if successor(k) fails, the other successors are still available for answering lookups for k. using a list of successors and applying replications of data items among them increases the robustness against very high degrees of node dynamics [4, 5].

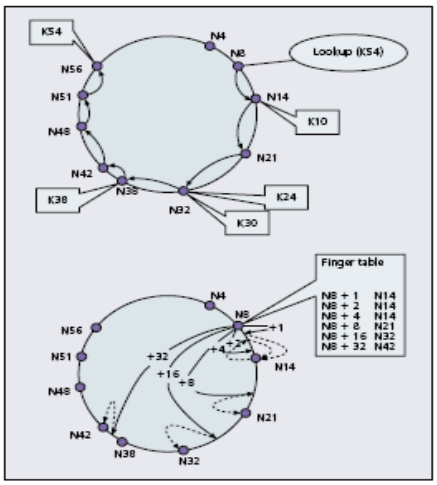

Figure 2. Chord ring [5].

3. Related Work

The following subsections are studies aim to improve the original Chord network by applying advanced techniques and algorithms to enhance one aspect or criteria of the original Chord. Also discussed in this section, several real deployment of chord into real networks implementations because of its magnificent characteristics.

3.1 Improvement on Chord to Achieve Better Routing Efficiency

Chord has often been known by its lack to routing locality. Even if the object is near the source of a query, it is often the case that one or more hops through the overlay will be needed for the object to be found.

A routing efficiency enhancement protocol is proposed in [6], called PChord, to solve this problem. PChord aims to achieve better routing efficiency than Chord by making use of proximity of the underlying network topology. Because the node that has sufficient routing information could reach data with a simple direct hop through IP and no need for extra overlay hops

Routing efficiency could be measured by Relative Delay Penalty (RDP) of the query. RDP is defined as the ratio of the distance a query travels through the overlay network to an object and the minimal distance to that object. Original Chord shows poor performance and routing efficiency under RDP because it does not consider network proximity at all [6].

Since data and services are replicated and could be transferred from one node to another, we need to find the nearest replica for the source query. The authors of [6] introduced the object pointer indirection layer of Tapestry into Chord overlay (i.e. the objects stored in the overlay are pointers to the location of the actual data) and trying to find ways to make object location in such an overlay efficient. To solve the routing problem under, a routing scheme combining proximity routing with basic routing algorithm of the original Chord is presented, which aims to achieve low RDP on Chord overlay. This routing efficiency enhancement overlay is called PChord [6].

The main contribution to Chord is to include a new proximity list into Chord's routing table. Proximity is weighed by RTT (Round Trip Time) which can be easily measured as the time it takes for a simple specified message to travel from one PChord node, across the network to another PChord node, and back [6].

An entry in the proximity list contains the IP and identifier of the proximate node. When a new PChord node joins the overlay, it have an empty proximity list. This PChord node will find some other PChord nodes near to it with RTT lower than certain predefined value through routing communication. It will add such kind of nodes to its proximity list. Meanwhile, these two PChord nodes will copy all different entries of the proximity list from each other. The length of proximity list will increase until the PChord node finds all PChord nodes in the same network partition which it belongs to.

The key adjustment of routing algorithm in PChord is the choosing of next hop. Next hop is not only selected by the entries in the finger table, but also selected by the entries in the proximity table [6].

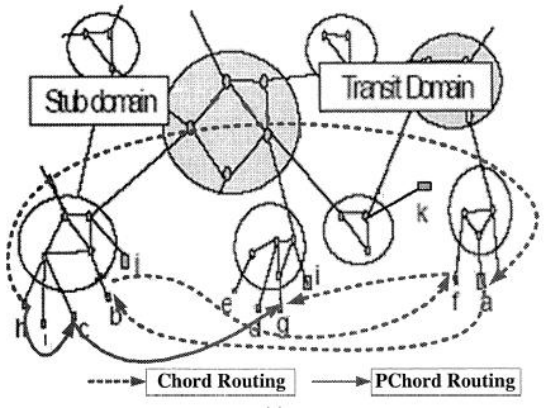

Figure 3. Routing examples of Chord an PChord [6].

Routing examples of Chord and PChord are illustrated in Figure 3, which node h is trying to locate node g. For standard Chord overlay, the hops of routing process is the dotted arrowhead line in Figure 3, which are h,a,b,f,g of 4 hops. For PChord, as node h holds a proximity list it will choose c as its next hop for this routing process, for c is closest preceding hop to the target g in key space from all h's local routing information. This hop of h is chosen from the entry of the proximity list. Then c will choose the next hop from its proximity list and finger list the same as h does. Node g is in the c's finger list, which will be chosen as next hop of c. This routing process on PChord is composed of only 2 hops [6].

Figure 4 illustrates the average RDP of PChord and Chord, it shows that it is lower for PChord than Chord when the distance between query source and target document is not long. When the distance is far away it shows similar performance since the real distance is physically far away not logically. It proves that PChord increase the routing efficiency by decreasing the hop numbers and limiting the routing path crossing the same network zone only once [6].

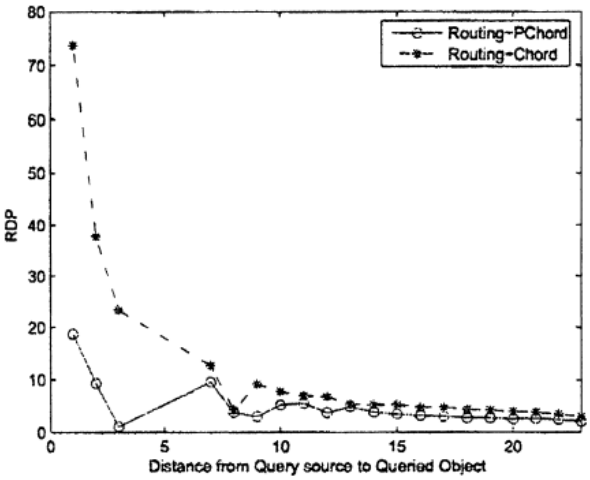

Figure 4. Routing RDP according to the distance from querying node to target document [6].

3.2 Improvement on Chord to Achieve Better Routing Security

Chord use DHT to allocate data items using lookup requests. With DHTs, each peer participates in bypassing the lookup requests to the correct destination and this routing must be conducted correctly to find the peer responsible for the requested file. Unlike IP routing, any peer can become a malicious router and easily corrupt the entire routing. Attackers may modify, drop or misroute lookup requests. The attackers can also deny the availability of certain data or provide manipulated data to other nodes. The attackers can even forward improper overlay route updates.

The solution to deal with these security concerns is proposed as an extension to Chord named Sechord in [7].

The main idea is that the source can determine whether the next hop is valid or invalid (i.e potential malicious node) by estimating how far the next hop is from its finger pointer. If the next hop is too far away from the finger pointer, the source can sense that malicious node might be encountered [7].

The modifications on original Chord is by forcing initially no trust between two nodes and each node make use of its locally available information to evaluate next hops during the lookup routing process for their validity [7].

Defense mechanism target three types of security threat [7]:

- A malicious node does not respond to lookup requests from other nodes.
- An attacker forwards a lookup request to some random node, not the correct next hop preventing the lookup request from reaching its final destination.
- A group of attackers collaborate to forward lookup requests to malicious node.

The attackers maintain two finger tables, the legitimate finger table that the attackers use for routing among themselves, the other one is the finger table that consists of the malicious node for each node at the legitimate finger table [7].

One important property in SeChord is that $|p-f| \leq |p-q|$ and $|q-f| \leq |p-q|$ where p and q are two consecutive peers, f is a finger pointer falls somewhere between p and q [7].

Modifications on chord are to store the identifiers of the successor and predecessor of each node in the finger table to compute the average numerical distance between two node identifiers [7].

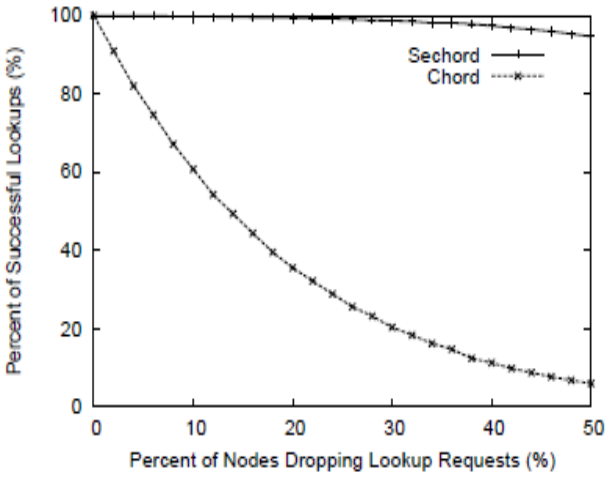

Figure 5. Lookup success ratio in the presence of nodes that randomly misroute lookup requests [7].

In Figure 5, the result shows that the lookup success percent of Sechord is higher than that of Chord when random routing threat is present. The lookup success percent of Chord decreases rapidly as more malicious nodes are present and perform misrouting. When 25% nodes are affected by attackers, the success ratio is reduced to 28%; with half of the nodes in chord network compromised, the success ratio becomes no higher than 10%. On the other hand, the success ratio of Sechord stays no lower than 90% for all percentages of compromised nodes [7].

3.3 Improvement on Chord to Achieve Better fault tolerance

Peer-to-peer systems are subjects to failures due to the following reasons [8]:

Global: Peer-to-peer systems should be implemented to function on a global region.

IP: Peer-to-peer systems typically organize into overlay networks on top of IP.

Self-organizing: These systems should be truly distributed systems.

Scale: A peer-to-peer system should be highly scalable.

Unreliable Nodes: Nodes in a peer-to-peer system should be considered highly unreliable due to the present of malicious nodes.

The solution is to make one particular peer-to-peer system, Chord, resilient to random node failures is done by repairing incorrect routing state by periodically refreshing all routing connections in each node. This prevents faults by removing connections to dead nodes, damaged links, and outdated links [8].

There are two mechanisms added to Chord to make it fault tolerant. First, redundancy allows Chord to handle many individual node failures without the entire system crash. Second, an aggressive repair algorithm fixes any links that may be broken or outdated [8].

Original Chord stated that so long as each node knew its successor, a lookup would return the correct node. There is a weakness in this scenario, if just one node fails, this condition no longer applicable.

The solution is to maintain links to r successors, not just one. If the successor of a node fails, the node can try its backups [8].

Theorem [8]: If each node has r successors, and each node fails independently with probability p, then the reliability of Chord is least $1- N p^{r}$.

Given that Chord can handle painful failures of nodes; we must repair the broken links before the system be vulnerable to future failures. Because when nodes have broken links, they are vulnerable to additional failures. Even if a broken link has not been detected yet, it should still be repaired. Otherwise, potential faults might arise leading to future problems. Broken fingers degrade the lookup performance. The sooner they are fixed, the faster lookups will be. Trying to allocate broken links during a lookup cause latency to that lookup. But it is better to discover and repair the broken links outside critical path [8].

The repair mechanism runs in a background thread periodically, once every 60 seconds. Every successor link and every finger entry is updated. Unfortunately, the repair mechanism adds overhead to Chord. But repairing broken fingers reduces lookup latency since no time for lookup failure is wasted and this balances some of the overhead [8].

In figure 6, the probability of failure is varied to specify how many failed nodes Chord can handle. The experiment examines using 5, 15, and 25 successors. The breaking points are the important points of this graph. For 5 successors, reliability drops after just 5% of the nodes fail. For 15 successors, it takes a 40% failure before the system crashes. For 25 successors, the breaking point occurs near the 2/3 failure rate. This is predictable results since the recommended number of successors for a 10, 000 node system is 2 log $10{,}000 \approx 26$ [8].

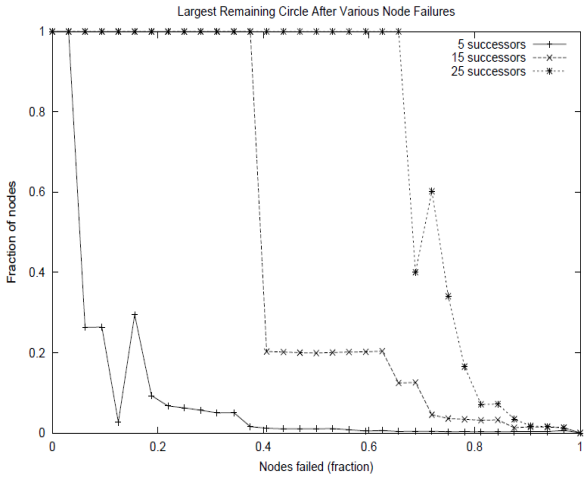

Figure 6. Largest remaining Chord circle after a large number of simultaneous node failures [8].

3.4 Improvement on Chord to Achieve Better Lookup Accuracy and Performance in Mobile P2P Network

Chord was designed initially for wired networks. When used in wireless network environment, many new issues are introduced. For example, the nodes in an unstable wireless network usually leave or rejoin the P2P network easily and frequently. In this case, the routing information in every node must be updated, and since this update takes a mount of time, this may lead to lookup failures when the nodes retrieve this not updated routing information [9].

The solution is to keep Finger Table fresh and self-update. This proposed modified Chord protocol is called Mobile Robust-Chord (MR-Chord). MR-Chord improves the Chord networks lookup success rate, improves the overlay consistency, and minimizes the lookup delay time in wireless mobile [9].

MR-Chord protocol consists of the following steps: modification in finger fable, real-time fix scheme, and by-detect fix scheme.

The modified finger table adds three columns to save the lookup outcomes as shown in table 1 [9].

Definition of Succ field:

– If the lookup success via the finger[i] → Succ[i]=Succ[i]+1

Definition of Fail field:

– If the lookup fail via the finger[i] → Fail[i]=Fail[i]+1

Definition of WeakNode field:

Fail – Succ ≥ 2 → WeakNode=True

Otherwise WeakNode= False

Tabel 1

The Modified Finger Table

Finger	Successor	Succ	Fail	WeakNode
N42+1	N43	3	3	False
N42+2	N47	4	0	False
N42+4	N47	5	1	False
N42+8	N51	2	1	False
N42+16	N0	1	2	False
N42+32	N8	2	6	True

The Real-Time fix scheme can fix the false finger entry in the finger table when a lookup fail happens. A node n starts a key lookup in the original Chord lookup mechanism the finger node p is broken. The information of the broken finger node p is in node n's finger table. Replace this bad finger entry by copying the previous finger entry to it. The node n tell its successor and predecessor about the information needed to fix their finger tables because the two node's finger tables usually have the same finger entry. On performing the next same lookup, we can use the fixed finger entry before the finger table traditional update, as shown in figure 7 below [9].

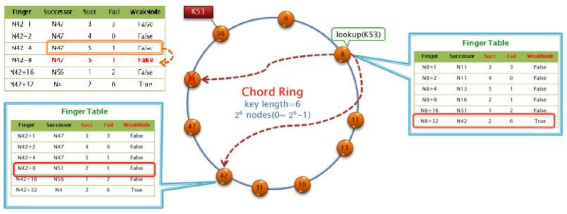

Figure 7. An example for Real-Time fix scheme [9].

The By-Detect fix scheme uses statistics of lookups to detect the finger nodes. If " Fail-Succ" > 2 then "Weaknode= True" and we call it weak node. When the finger node is set as a weak node, the weak finger starts to check if the finger nodes in the finger table are alive or not and fix the error finger with those bad nodes. This will predict bad fingers before even any lookup fail [9].

An example, when node 8 gets a lookup failure, it records the failure and checks whether"Fail"-"Succ" > 2. If this is true, node 8 asks node 42 to start the check procedure until"Succ-Fail" >2. In the check procedure, node 42 checks its finger nodes in its finger table [9].

Lookup Success Rate in MR-Chord is higher as shown in figure 8. MR-Chord keeps the accuracy of finger tables. If a lookup request is forwarded via all correct and alive successors, it can make a success lookup [9].

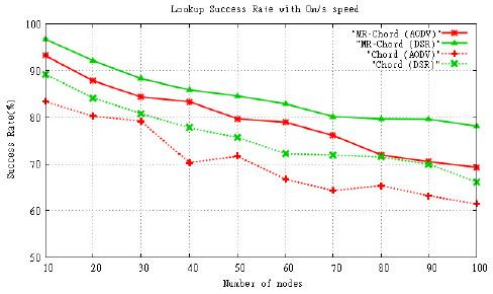

Figure 8. The Lookup Success Rate with fixed network [9].

Lookup Delay Time in MR-Chord is lower as shown in figure 9. The reason behind this is the fact that MRChord protocol is capable of fixing the bad entry in the finger table. It allows the lookups to get fewer bad successors and to find the resource quickly [9].

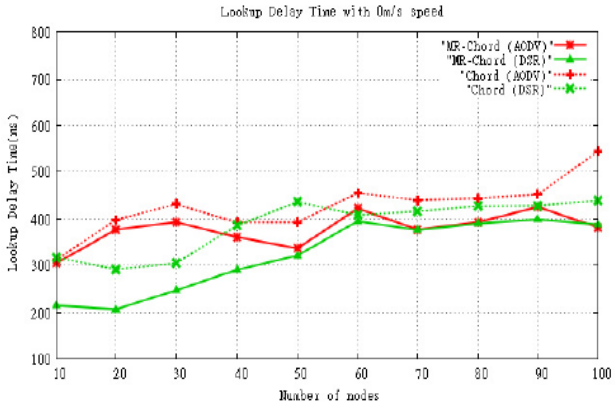

Figure 9. The Lookup Delay Time with fixed network [9].

3.5 Building Power Grid Applications using Chord

The power grid transfer and distributes electricity from the power factories to the consumers. Improving performance of power grid (optimize power quality, cost, energy loss, etc.) needs accurate management and distributed control. The dynamic environment in grid applications requires updating the configuration of the information infrastructure beneath the power grid periodically and at run time not only static configuration. Also a geographically distributed nature of power grid applications adds new needs to self updating network [10].

A great solution is to use Chord network since it is a purely decentralized peer-to-peer network that show major advantages for power grid applications from reliability, scalability, availability, dynamicity and quick search, an example is shown in figure 10.

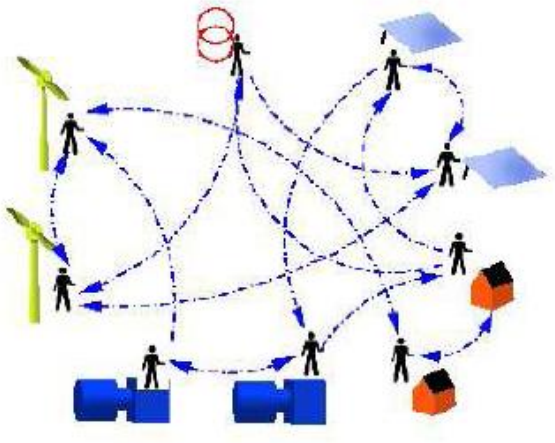

Figure 10. Example of P2P network that applied to power grid [10].

Today, there is an important trend to use small Distributed Generation (DG) in low or medium voltage [10].

Control of power elements in power grid has three control levels [10]:

- Primary control is used to balance both active and reactive power, by measuring frequency locally.
- Secondary control is basically used for maintaining rated voltage levels or rated frequency and scheduled power transfers
- Tertiary control utilizes generators output for economic reasons.

The last two control levels both require some form of collaboration and communication with other generator controllers. When adding DGs to the power grid for these levels of control, centralized control systems are not suitable because they are expensive since you will need dedicated communication lines and a large number of load balancing servers. So we are looking for less expensive infrastructure for these control paradigms [10]. Chord network meets these requirements and can solve them.

Power grid components (e.g. generators, dispatching loads) are connected to an autonomous control entity (agent). An important aspect of agents, besides autonomy, is that they communicate with a society of (similar) agents, from which they may explore external information of their interest. These societies can be built easily by setting up a peer to- peer network [10].

The main advantages of Chord network for power girds are efficient lookup search, decentralization, self organization, and no partitioning.

Efficient lookup search: Using a Chord network, all agents' communication can be done efficiently and quickly. (log N) steps is needed (N is the number of agents) [10].

Decentralization: Power grid agents act as peers in Chord network. Peer has capabilities or privileges than any other [10].

Self-organization: as a result of decentralization, there is no central agent to control and coordinates the other agents with each other. Leaving and joining of agents does not dramatically reduce the system performance. [10].

No partitioning: By Chord network, active and alive agents always can communicate with each other without partitioning.

The dependability aspects of Chord network for power grid applications is considered with reference to reliability, scalability, and availability [10].

Reliability: Chord topology is purely decentralized and there is no single point of failure so there is no fear of having the system totally crashed because of single point failure [10].

Scalability: As Chord network has efficient lookup routing searches, increasing the number of lookups doesnt produce a huge overhead on the network communications [10].

Availability: Chord automatically updates its internal tables as a response to joined and failed nodes in the network [10].

3.6 Improvement on Chord to achieve better load balancing

Most of the work on Web services discovery depends on using centralized registries. Although they are effective they suffer from the traditional problems of centralized systems, which are primary performance bottlenecks and single points of failure [11].

The solution is to extend the Web services discovery scope using P2P technology. Chord is a good candidate because it has good scalability, robustness, load balancing and self-organization [11].

The research of Web service discovery can be divided into two steps: service discovery model and service discovery algorithm [11].

A kind of services discovery model named WSDBC (Web Service Discovery based on BalanceChord) is proposed in order to expand the Web services discovery region and increase discovery efficiency [11].

In order to achieve load balancing between different nodes in WSDBC model, node join-in algorithm and self-balancing algorithm are proposed.

In BalanceChord the node ID is created by the code of NAICS instead of hashing information of the node itself and object Key, see figure 11 [11].

NAICS (North American Industry Classification System) is used to classify the service.

Service can be defined as an ordered 3-tuple [11]: < SN,CN,C > where

- SN represents the service name
- CN represents the category name
- C represents category code

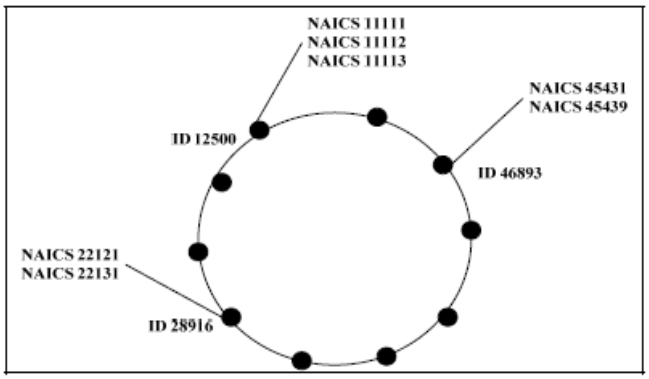

Figure 11 An Example of BalanceChord [11].

The load of node is the summation of invoking times of all services in the node. It can be defined as follows:

$$L_{node} = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}$$

Where node L represents the load of node, m represents the numbers of service category in the node, n represents the numbers of service in a category. p ij represents the times which the service is invoked. At the beginning, L node=0. As time passing by, the load of nodes will be different [11].

In order to improve the load balancing of nodes in the WSDBC model, two kinds of strategy are proposed: New node joins beside the node which is overloaded in order to share the load (node join-in algorithm). The load of nodes can be automatic self balancing (self-balancing algorithm) [11].

New Node Join-in Algorithm [11]:

Step 1. New node applies to join in;

Step 2. Searching for the node with greatest load;

Step 3. New joined node sets its ID;

Step 4. Service information transfer to new joined node.

Self-balancing Algorithm [11]:

Step 1. Comparing load between the node and its successor;

Step 2. If the node is overloaded, go to Step 3; else go to Step 1;

Step 3. Node sets its new ID;

Step 4. Service information transfer to its successor.

Figure 12 shows the load information of Chord; Figure 13 shows the load information of BalanceChord. From the figures we can see that the load information of nodes in Chord is very unbalanced. The load information of some nodes exceeds 100 and some near to 0. Compared to Chord, BalanceChord has better load balancing [11].

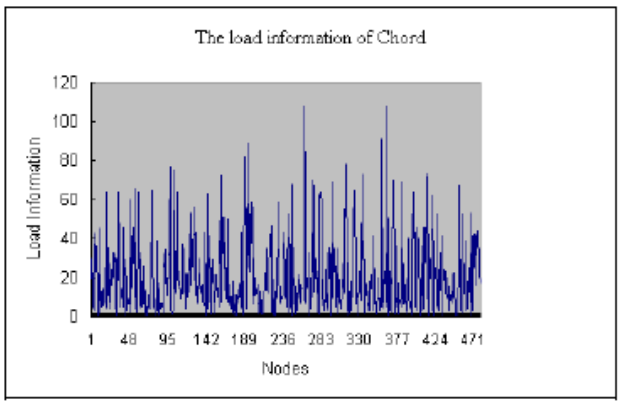

Figure 12 The Load Information of Chord [11].

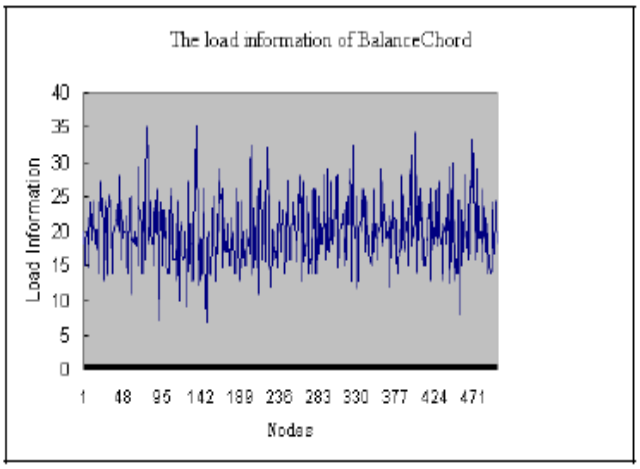

Figure 13 The Load Information of BalanceChord [11].

4. REFERENCES

[1]. Luc Onana Alima, Ali Ghodsi, and Seif Haridi. . "A framework for structured peer-to-peer overlay networks," In LNCS volume of the post-proceedings of the Global Computing 2004 workshop. Springer-Verlag, 2004.

[2]. Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. "One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks," In Proceedings of the SIGOPS European Workshop, Saint-Emilion, France, September 2002.

[3]. H. Balakrishnan, F. Kaashoek, D. Karger, R. Morris, I. Stoica, "Looking Up Data in P2P Systems," Communications of the ACM, V. 46, N. 2, (February 2003).

[4]. Stoica, R. Morris et al., "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," IEEE/ACM Trans. Net., vol. 11, no. 1, 2003, pp. 17–32.

[5]. Eng Keong Lua, Jon Crowcroft., " A Survey And Comparison Of Peer-To-Peer Overlay Network Schemes," IEEE Communications Surveys & Tutorials,Second Quarter 2005

[6]. Feng Hong, Minglu Li, Min-you Wu, Jiadi Yu, "PChord: Improvement on Chord to Achieve Better Routing Efficiency by Exploiting Proximity", IEICE Transactions on Information and Systems, E89-D(2), pages 546-554, 2006.

[7]. Keith Needels, Minseok Kwon., "Secure Routing in PeertoPeer Distributed Hash Tables," ACM 9781605581668/09/03, 2009

[8]. Daniel Adkins., "Making Chord Robust," University of California, Berkeley, 2012

[9]. Jian-Ming Chang, Yi-Hsuan Lin, Isaac Woungang, Han-Chieh Chao, "MR-Chord: A Scheme for Enhancing Chord Lookup Accuracy and Performance in Mobile P2P Networks", (Accepted Jan. 9, 2012) to appear in Proc. of the IEEE Intl. Conference on Communications (ICC 2012), Ottawa, Canada, June 10-15, 2012.

[10]. Beitollahi, H., Deconinck, G, "Analyzing the Chord Peer-to-Peer Network for Power Grid Applications", In: Fourth IEEE Young Researchers Symposium, 2008.

[11]. Li CHEN, Zilin SONG, Shiming ZHENG, Wenjie SUN, Zhanfeng WANG1, " A Model of Web Service Discovery Based on BalanceChord", Journal of Computational Information Systems 7: 7 (2011) 2241-2247, 2011

[12]. Structured P2P Overlays ppt presentation of Dr.-Ing. Kalman Graffi, www.p2pframework.com

[13]. Z. L. Kis and R. Szabó, "Interconnected Chord-Rings," Network Protocols and Algorithms (NPA), vol. 2, iss. 2, pp. 132–146, 2010.

[14]. ZENG Xiao-yun," Hybrid P2P Model Based on Chord Protocol", Tsinghua Tongfang Knowledge Network Technology Co., td.(Beijing)(TTKN, 2010