ChordX – An Improved Bidirectional Chord protocol with network locality awareness

Jeris Alan Jawahar
North Carolina State University
Department of Computer Science
jjawaha@ncsu.edu

Sharmin Lalani
North Carolina State University
Department of Computer Science
slalani@ncsu.edu

Durgesh Kumar Gupta
North Carolina State University
Department of Computer Science
dgupta9@ncsu.edu

ABSTRACT

Chord is a distributed peer to peer lookup protocol that is used to efficiently identify the node that stores a data item. It has gained widespread adoption primarily due to some of its features like simplicity, scalability and adaptability to multiple concurrent node joins and departures. The original Chord protocol relies on the information present in the finger table for routing, however the finger table does not cover, or represent the entire Chord ring; rather in a dense network the finger table represents only one half of the Chord ring. Another shortcoming of the original design is that it does not take advantage of the underlying network topology or node proximity. This leads to increased routing latency and poor bandwidth usage even when nodes are geographically and physically close. In this project, we have implemented the Chord protocol and addressed both the above cited problems, firstly by optimizing the finger table design to ensure better coverage of the Chord Ring, and secondly by assigning network locality aware identifiers to new nodes that join the Chord network.

CCS CONCEPTS

• **Computer Communication Networks** → Distributed Systems

KEYWORDS

Distributed Systems, Peer to Peer, Chord, Distributed Hash Tables (DHTs), Proximity Routing, Topology Awareness

1 INTRODUCTION

Structured P2P Systems have gained a lot of traction in the last decade for developing many applications related to cooperative storage, content distribution, group communication, etc. due to their scalability, fault tolerance and self-organization. With the introduction of Distributed Hash Tables in the first half of the previous decade, many protocols for locating a data item or a resource within structured P2P systems have been designed, with the most notable among them being Chord [1], Pastry [2], Tapestry [3], etc.

For many Peer-to-peer systems, it is desirable that the process of locating the node for a data item be fast, highly scalable, and adaptive (so that nodes can join in an ad-hoc fashion), and robust (so that multiple, concurrent node failures do not hinder the ability to locate data). Chord encompasses all these features. It uses consistent hashing to assign m-bit identifiers to nodes and keys, thus mapping them to an ordered logical ring, such that the node immediately following a key in the clockwise direction is responsible for that key. When a single node joins or leaves the ring, key relocations are limited to that node's immediate region of the ring. To accomplish fast lookups, each node maintains a *finger table* that stores pointers to nodes at exponentially increasing clockwise distance from itself. The lookup algorithm is similar to binary search, thus working in logarithmic time. To locate a node responsible for a given key, the source node picks the closest predecessor node for that key in its finger table and repeats this process at that node and so on, thus halving the distance to the target node in each iteration.

One of the biggest flaws in Chord is that node joins do not take into account the overlay network topology. There is no correlation between the distance of node identifiers on the logical ring and the physical distance of nodes in the underlying network. The protocol measures lookup performance by the number of logical hops, overlooking the geographical distance and link speed between them, which can lead to long physical routes even for close logical hops. This can severely impact the routing latency.

Another issue is the finger table design. For keys that are located near the predecessors of the source node, the lookup messages need to traverse almost the entire Chord ring to reach the target node. This can be prevented by maintaining finger table entries for the anticlockwise direction as well.

We propose ChordX, a modified version of the Chord protocol. Our contributions are as follows.

- ChordX exploits the overlay network topology by dividing the Chord ring into geographically distributed zones. A new node can be explicitly added to a zone in which other physically close nodes are present. We also provide a robust algorithm that provides the same functionality implicitly. The latter approach assigns identifiers to a new node from a pool of random identifiers from each zone, based on the latency between the probable successor and predecessor nodes for each random identifier.
- The finger table size has been increased from m entries (where m is the node identifier bit size), to $2m$-1 entries, to include successors in clockwise as well as anti-clockwise direction, and provide better ring coverage.
- Dead node detection corrects invalid successor pointers after node crashes, to prevent query drops and restore the ring to a consistent state.

Because of these improvements, the average distance that must be traversed by a message exchanged between two nodes is reduced significantly. The node join time has increased on an average by

25%, but the lookup latency has been improved by around 12%. In other words, ChordX sacrifices join time to achieve better routing efficiency. This model is best suited for applications where the network churn is relatively low compared to the volume of key lookups.

The rest of the paper is organized as follows: Section 2 throws light on the detailed design of the improved Chord protocol with emphasis on various nifty improvisations. Section 3 deals with experimental evaluation and setup along with detailed analysis of the results obtained. Section 4 discusses some of the previous works that have been undertaken with respect to network proximity awareness and Finger Table optimizations. Finally, in Section 5, some limitations of our implementation are presented with concluding remarks following suit in Section 6.

2 SYSTEM DESIGN

2.1 ChordX Protocol Overview

ChordX is a wrapper over the Chord protocol that aims to improve its routing efficiency. Figure 1 shows the architecture diagram of the ChordX protocol.

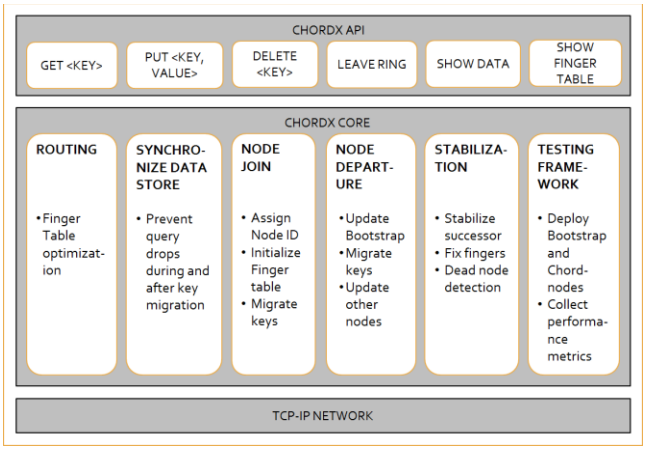


Figure 1: ChordX Architecture

The rest of the section discusses in detail the improvements made to the original Chord protocol.

2.2 Node Joins

ChordX has been designed to deal with multiple nodes joining the system, or leaving the system either due to a crash or voluntarily, at any moment of time. This section describes how ChordX deals with these situations and how network proximity awareness has been embedded into the protocol using two different approaches.

A ChordX Node adds itself to the ring, by invoking the Bootstrap server through a remote method. The IP address of the Bootstrap server is specified as a command line argument when creating a new ChordX instance. When the Bootstrap server receives a request for a new node join, it assigns an m-bit identifier using SHA-1 as the base hash function. A node identifier is chosen by hashing the node's IP Address and a unique Java RMI handle identifier which indicates the Chord Node instance number running on a system. To prevent collision when hashing, we have added the current timestamp also to calculate the identifier.

The original Chord protocol measures efficiency in locating objects in terms of logical hops. It does not exploit network proximity in the underlying physical network. This leads to increased latency when searching for a target that is physically close, but may be logically far in the Chord ring. To enhance the original protocol with proximity awareness, we have proposed and implemented two approaches that we discuss below. The first method uses a Topology Aware Zone selection mechanism to exploit the locality property whereas the second method uses a proximity based identifier assignment mechanism to assign a unique node identifier to the new ChordX node. Both the above approaches divide the Chord ring into *m* different geographical areas or zones.

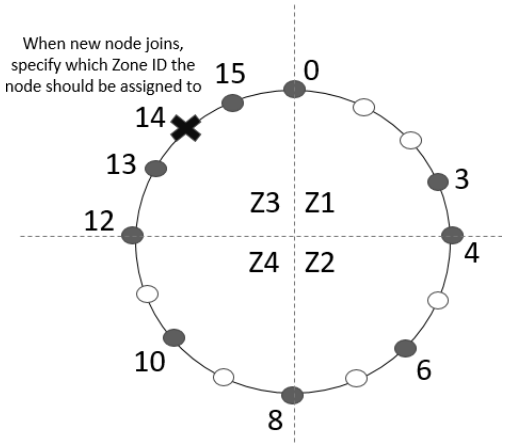


Figure 1: Topology Aware Zone Selection

2.2.1 Topology Aware Zone Selection – Dividing the Chord ring into multiple geographical zones enables nodes that are physically close to be placed logically close by explicitly specifying the zone they should be a part of. As the administrator of any network would be aware of the network topology, he can specify the zone in which the ChordX node should be added. Thus, proximate nodes can fall within the same zone, thus reducing the network latency when routing messages. For any zone with identifier *i* in an *m*-bit Chord ring, the range for the zone is given by the formula,

$$\text{range} \in [(i\text{-}1)*m,\ i*m)$$

Suppose the Chord Ring is divided into 4 different geographical zones as shown in Figure 1, identified by Z1, Z2, Z3 and Z4 respectively. The colored circles represent nodes already in the ring and uncolored circles represent nodes yet to join. Let's say a new node indicated by the cross sign wants to join the network. Based on the geographical region this new node lies in, the network administrator can specify the zone ID for it. The Bootstrap server will then compute and assign a unique *m*-bit identifier using the hash function such that it is within the zone range. Suppose the specified zone is full, then the Bootstrap server will randomly assign a node identifier from another zone.

2.2.2 Proximity Based Identifier Assignment - In this method too, we follow the same idea of dividing the Chord ring into *m* different zones. Once a new node requests to join the Chord ring, instead of generating only one identifier, we generate a set of *m* unique identifiers. For each of the random identifiers generated, the Bootstrap node returns the probable successor and predecessor node. The new node then measures the latency to the probable predecessor and successor node for each of them by calculating the time taken for a remote method invocation. It then selects the identifier, for which latency is the least.

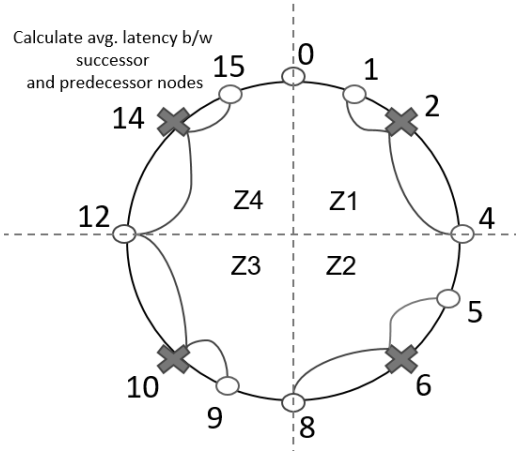


Figure 2: Proximity Based Identifier Assignment

In Figure 2, the nodes already present in the Chord ring are represented as empty circles. The crosses in the figure depict the *m* random identifiers generated by the Bootstrap for the new node. For node identifier 2, the successor and predecessor are 1 and 4 respectively. Similarly, for node identifier 6, it is 5 and 8, for node identifier 10, it is 9 and 12 and lastly for node identifier 14 it is 12 and 15 respectively. If any zone if full, then we do not generate a random identifier falling within that zone. The pseudocode of the above method is put forth in Figure 3.

Algorithm - Proximity based Identifier assignment

```
Extract the new node information like IP Address and
Port
for each unfilled zone:
    1. Generate a random node ID ni for that zone
    2. Identify the successor and predecessor for
       ni
end for
for each random node ID generated:
    1. Calculate latency to successor and predecessor
       node
end for
Identify node identifier with least latency
Assign identifier to new node
```

Figure 3: Algorithm for Proximity based identifier assignment

By using the latency metric, we get a good approximation of the fact that nodes that are physically or geographically close to one another in the real world will also be placed closely together in the Chord ring. Thus, efficiency in message routing can be obtained amongst the participating nodes.

2.3 Node Departures

When a node *n* voluntarily chooses to depart the network, it will transfer all its keys to its immediate successor node *s*. Further, it will also notify its successor *s* and predecessor *p* before leaving. Node *p* will remove node *n* from its successor list and add the successor of node *n* as its new successor.

Once the successor and predecessor node pointers are correctly updated, node n sends a remove from ring request to the Bootstrap server and updates the finger table entries of other nodes to that point to it.

2.4 Routing

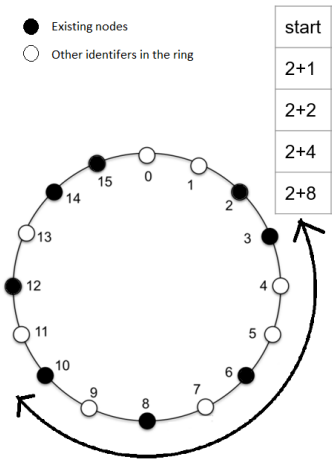
Each node stores a finger table of size *m*, where *m* is the node identifier bit size. A finger table entry *i* is a <start, successor> pair; start is calculated as $[(id + 2i\text{-}1) \bmod 2^m]$, where *id* is that node's identifier, and successor is the identifier of the node following the start value in the clockwise direction. In the original protocol, the finger table only represents one half of the Chord Ring. As seen in Figure 4, the finger table entries for node 2 are in the range [3, 10], while the range [11, 2] is not represented. This means that keys closer to the node in the anticlockwise direction take more hops to reach. For example, consider a lookup request for key 15 from source node 2. The closest preceding node in the finger table of node 2 is node 10, thus it will look for the next closest preceding node in the finger table of node 10. This process repeats till the successor for key 15 is found. In our example, this chain of nodes includes nodes 10, 14, and 15.

2.4.1 Finger Table Optimization - We have increased the finger table size from *m* to *2m-1* entries to incorporate anti-clockwise entries as well. The first *m* entries are calculated the same as above. For the last m-1 entries, start is computed as $[(id + m - (2i\text{-}1)) \bmod 2^m]$. Continuing the same example, we can see from Figure 5, that the chain of nodes contacted now by node 2 to find the successor for key 15, includes 14 and 15. For larger rings, there will be a greater reduction in the number of hops.

There is still another issue with the finger table structure that we are not tackling, that is redundant finger table entries. This is a bigger problem in sparse rings, as denser rings will have fewer redundant entries. Also, we note that if the distribution of nodes in the ring is almost uniform, then redundancy is more likely to be found in the first few entries. We make this observation from the fact that the distance between consecutive start values doubles as we move down the finger table.

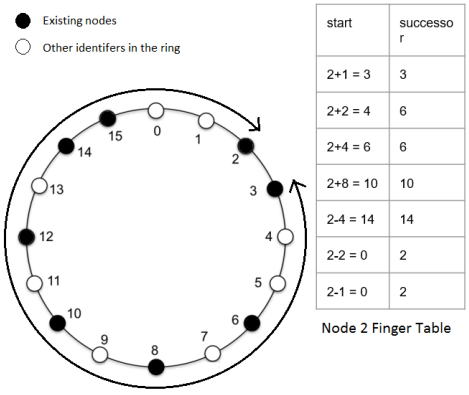
Removing redundant entries will save some memory, but at the expense of algorithm complexity. As the network topology changes over time due to node joins and departures, the entries which were redundant before may not be redundant later. These entries, if eliminated before, may need to be added again to ensure good ring coverage. Determining when to add back the eliminated entries is non-trivial, and it will make the periodic stabilization

routines complex. We are not dealing with redundancy, as the memory savings are an overkill compared to the overhead. One way to reduce the time taken in traversing the finger table during lookups, is to add pointers to the next unique entry in the table. This can allow us to skip the duplicate entries.



start	successor
2+1	3
2+2	6
2+4	6
2+8	10

Figure 4: The finger table in the original protocol only represents one half of the Chord Ring.



start	successor
2+1 = 3	3
2+2 = 4	6
2+4 = 6	6
2+8 = 10	10
2-4 = 14	14
2-2 = 0	2
2-1 = 0	2

Node 2 Finger Table

Figure 5: Finger table with bidirectional entries

2.5 Stabilization

At periodic time intervals, each node sends heartbeat messages to its successor to verify if it is alive. If the successor is alive, the node checks that it is a valid successor. This ensures that all the nodes in the ring always have correct successor pointers, even if some update operations fail during network churn. Additionally, the finger table entries are also recomputed, to accommodate node joins and departures.

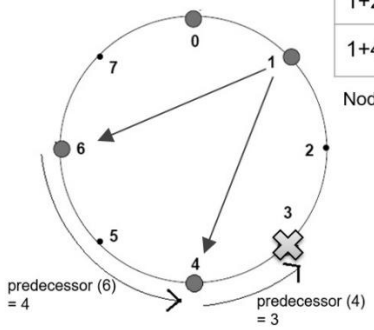
2.5.1 Dead Node Detection **-** After two unsuccessful connection attempts, a node proclaims its successor to be dead. It updates the bootstrap server, and other chord nodes eventually learn about it through their periodic fix fingers routine. The node then looks for the next probable successor in its finger table. If finger table does not contain information about any other node, it contacts the bootstrap to assign a new successor. The algorithm below explains the process, and Figure 7 illustrates it.

Algorithm - Dead Node Detection

```
for each finger table entry i:
        if i!=my-node-ID and i!=original-successor-id:
                new-successor = i
                break
        end if
end for

if new-successor is set:
        while predecessor of new-successor!=original-
        successor:
                new-successor = predecessor of new-
                successor
        end while
        Set successor entry as new-successor.
end if
else
        Update bootstrap about dead node.
        Get new successor from bootstrap server.
end else
```

Figure 6: Algorithm for dead node detection



start	successor
1+1	3
1+2	3
1+4	**6**

Node 1 Finger Table

Figure 7: After Node 3 crashes, Node 1 follows the predecessor chain from node 6 to set node 4 as its new successor.

2.6 Key Migration and Data Store Synchronization

Keys are migrated to a newly joined node from its successor, and from a voluntarily departing node to its successor. To ensure that queries are routed correctly during and after migration, we need synchronization policies. We have used read-write locks for all operations that access the local data store (insert key, retrieve key, delete key, and migrate key).

A new node that joins the ring first relocates the keys that should be stored with it from its successor to itself, and then updates its neighbors. Until the neighbors are informed, the ring does not have knowledge of the new node, and no queries can be routed to

it. The locks prevent query drops on the old node during key migration. The queries are put on a hold till migration completes. These queued queries, and stale queries that reach the old node after key migration, are re-routed to the new node. This ensures that retrieve operations do not fail, and insert operations do not store the key with the wrong node.

3 EXPERIMENTAL EVALUATION

We have implemented the original Chord protocol in Java, and implemented ChordX on top of it. We evaluated ChordX and compared the performance with the original protocol on Virtual Computing Lab (VCL) [15].

3.1 Experimental Setup

Our experiment was conducted on virtual machines deployed in NCSU Virtual Computing Lab (VCL) where multiple virtual machines with same configurations [16] such as CPU, RAM, disk space etc. can be easily spawned. All the virtual machines used had one 64-bit CPU core, 2GB RAM, and 30GB disk space.

To test our protocol on geographically distributed nodes, we have used VMs in NCSU on-campus datacenter and Research Triangle Park datacenter. The average latency between VMs inside the same datacenter is 0.25ms and between the two datacenters is 7.8ms.

The ChordX instances have been developed in JAVA version 8. We have used the RMI library to achieve remote procedure calls. Using RMI library over sockets was preferred because it handles all the low-level connection management with additional error handling mechanism.

We have 3 components to run and test our implementations: (1) Bootstrap, (2) ChordX instance, and (3) Deployer. Bootstrap is responsible for adding new nodes into the Chord ring, as described in section 2.2. The testing module has been developed on the bootstrap. Deployer is a shell script that sets up, compiles and launches the ChordX instances on VMs. It takes a list of IP addresses, copies the source code to them via SCP using key-based authentication, and brings up a ChordX instance on each VM.

After the deployer has finished executing, the testing module performs key insert and retrieve operations on the chord ring, and collects performance metrics for all the operations. Key-value pairs are randomly generated, and a random node is picked from the list of running nodes as the source node for key insert and retrieve operations. The number of key-related operations to be performed can be configured in the script. We have tested with number of keys ranging from 1,000 to 10,000. To ensure concurrent key-related operations throughout the ring, and evaluate the performance under a more realistic workload, we have used multiple threads to perform these operations. The performance metrics we have used are average latency and average number of messages exchanged per operation. Each node records the start and end time for different types of operations and stores the difference locally. After all the keys operations have completed, the testing module running on the bootstrap contacts each node, retrieves the metrics and computes an average value.

3.2 Results and Analysis

In this section, we evaluate the ChordX protocol by implementing and deploying the service in a distributed environment setting. We compare the results obtained from the original and the improved protocol.

3.2.1 Node Joins

For measuring performance with respect to node joins, we calculate the latency measure in milliseconds. The latency is the difference in time from when the new node sends a join request to the Bootstrap server till it sends an acknowledgement of the successful join. This period also includes updating neighbors and finger table initialization. We evaluated the setup by increasing the number of nodes by 4 for each successive round of testing. We have observed that with the network proximity optimizations, there has been a significant jump in the join time. As the number of nodes in the ring grows, this time difference between the original and improved protocol increases. On an average the latency increased by 34% when a node joins the network using the Proximity based identifier assignment method. This increase in the metrics can be attributed to two main reasons. Firstly, the increase in finger table size means a new node needs to reach out to more nodes in the network to initialize its finger table. Secondly, there is added overhead involved in calculating the latency values for different probable successor and predecessor nodes.

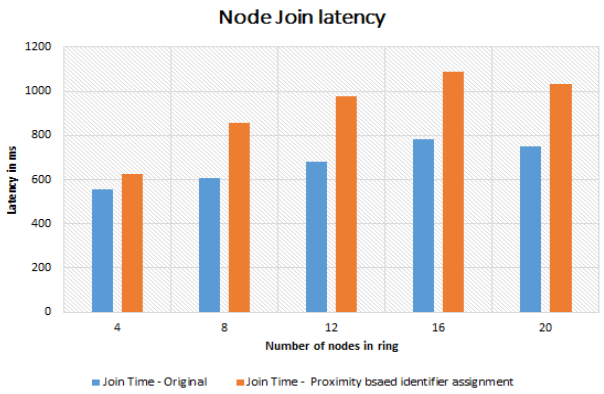


Figure 8: Latency measure for node joins

3.2.2 Lookup Operations - The performance measure for different lookup operations like insert and get key were obtained by averaging the metrics over 10,000 and 3,000 operations respectively. In addition to the latency measure in milliseconds, we have considered the hop count or the number of messages exchanged between different nodes to get relative measure of performance improvement. This new measure has been added to demonstrate that a node has a much better picture of the entire Chord ring after adding more finger table entries. Hence a node can directly reach out to a much larger set of nodes without having to route messages to more intermediate nodes before the destination is reached. For each number of nodes shown in the

figures below, we randomly pick nodes and perform concurrent key operations to get the metrics for each type of key operations. Figure 9 and 10 plots the average latency for both insertion and retrieve key operation. From these figures, it is evident that the latency values are less for the improved protocol. We see roughly a 12% performance improvement for the insert operation while for the get operation it is about 11%.

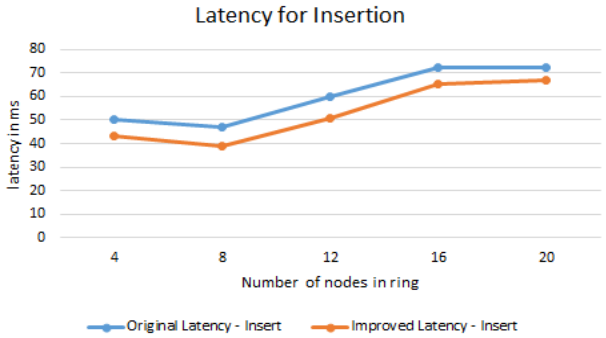


Figure 9: Latency for Insert Key operations

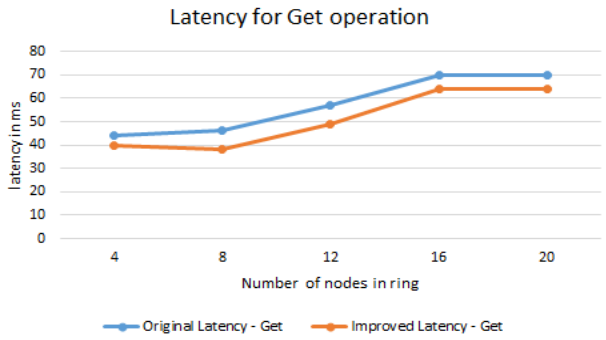


Figure 10: Latency for Get Key operations

The original chord paper [1] shows that lookup requires O (log N) message exchanges. Figures 11 and 12 show a plot of the number of hops involved or the number of communication messages exchanged for the operations. The values we have observed are in agreement with the theoretical bounds. Also, we have observed a marked improvement in the number of hops. The lesson that we can take from these findings is that the lookup latency grows slowly with gradual increase in the number of nodes, which confirms and demonstrates the scalability.

4 RELATED WORK

Several methods have been proposed for finger table optimization and enabling the nodes it to be topologically aware of its network. A bidirectional Chord system has already been suggested by Yan et al [4] which demonstrates the performance of a base-k finger table with different values of k, to optimize routing efficiency.

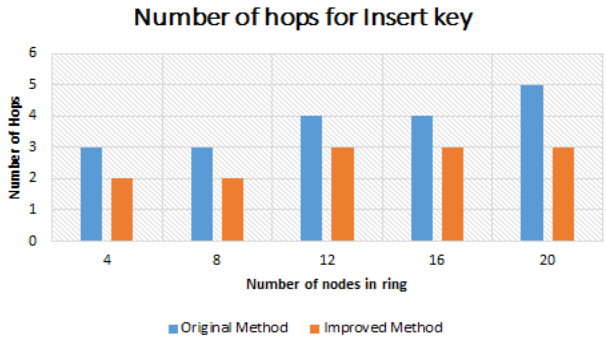


Figure 11: Number of hops for Insert Key Operation

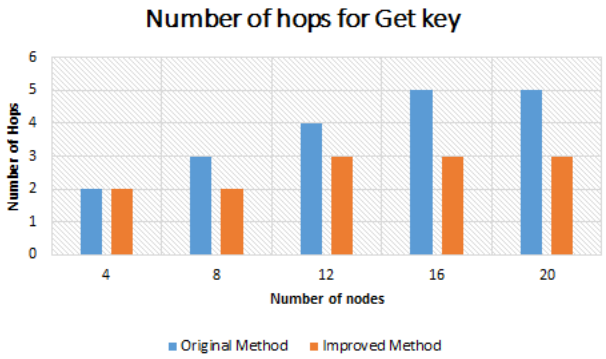


Figure 12: Number of hops for Get Key Operation

Bidirectional changes combined with Cache entry record and Semi-Recursive routing proposed in [5] improve the routing efficiency in P2P SIP based systems. The Chord topology is modelled as a bidirectional graph in [7]. A reverse link is added for each original unidirectional link and such a pair of symmetrical links is maintained by a single heart-beat message. Each node thus maintains a finger table and a reverse finger table. [6], [8], [9], [10] focus on replacing the duplicate entries in the finger table with nodes outside of the original finger table range, some of them needing modification to the original routing algorithm as well. These papers discuss the benefits in terms of memory optimization and routing latency, but do not throw light on how their approach performs under network churn.

Assignment of nodes identifier in an overlay network based on topology such that neighboring nodes in a physical network have close key spaces has been successfully used in CAN [11]. However, non-uniform distribution of keys results in load balancing problems. Also, mapping can be severely constrained when using one dimensional key space. For prefix based routing algorithms such as Pastry [2] and Tapestry [3], entries in the routing table are topologically determined which can minimize the distance a query needs to travel in the underlying physical network. Several changes to tapestry have been proposed in [12], but these do not decrease total communication required to create the routing table. In Pastry, the routing table is constructed in O

(log N) but it cannot be guaranteed that the entries are close as compared to Tapestry.

PChord [13] maintains a proximity list for each node in the network, which has a list of neighboring nodes that are in its close geographical range. In this protocol, some finger table entries are replaced with appropriate entries of proximity list; therefore, routing is performed among the nodes which are physically closer. One issue with this protocol is its inefficiency in the event of high churn, which increases overhead. In Chord6 [14], the hierarchical structure of IPv6 addresses is taken advantage of. The hash function generates identifiers of the nodes in a way that all nodes in the same domain have identifiers close to one another. The problem is that due to increasing number of Internet domains and low to average P2P network nodes that are physically in the same range, the efficiency of this protocol isn't optimal leading to its reduced usability. Fantar and Youssef in [17] proposed a new method to incorporate network locality by capturing and maintaining geometric coordinates which characterizes the network node locations in the Internet by using Global Network Positioning system. LDHT [18] is another work where instead of assigning random node identifiers as in traditional DHT algorithms, nodes are assigned locality-aware identifiers according to their Autonomous System Numbers (ASNs). As a result, a node will have more nearby neighbors than faraway neighbors in it the Chord overlay. This method has been found to be efficient in other DHT protocols like Symphony and Kademlia. In TAC [19], the authors introduce a local ring concept by dividing the geographical space into smaller areas. Nodes are added to a local ring based on their physical location. These smaller local rings then form a part of the larger Chord ring. Though it is efficient in terms of routing and bandwidth usage, it requires additional memory to store information about each node in the local ring. AChord [20] uses Anycast communication to ensure the nearest neighbor is fastest to respond to its join request and then constructs a neighbor table to enable faster and efficient routing. The only drawback of this method is that it requires the underlying network to support IPv6.

5 LIMITATIONS

Data replication - Keys are not mirrored at other nodes, thus it can cause data loss when any node fails. Also, the dead node detection algorithm declares a node as dead after two unsuccessful connection attempts. We have not changed the default connection timeout value of 15 seconds set by the JAVA RMI library. A node thus has twice the timeout duration to confirm that it is alive. If any node is not reachable for a duration more than that, it will be assumed dead, and the data stored with it will be lost.

Malicious node detection - Chord is not inherently resistant to malicious nodes that join the DHT. A malicious node can interfere with routing by dropping or modifying routing requests. It can also corrupt the finger tables of other nodes by advertising false routes. This can lead to failed lookups.

Routing with NAT - Requests can only be routed to public IP addresses. If a node with a private IP address connects to the internet through a NAT device that performs 1-to-N mapping, incoming connections can get dropped.

Ordering of operations - Key requests are not timestamped. Operations are performed in the order in which they are received, which is not always the order in which they are sent. Two insert operations for the same key that originate around the same time, face a race condition. Solving this problem needs clock synchronization.

6 CONCLUSION

In this paper/project we have implemented and evaluated an improved Chord protocol which exploits network proximity while assigning locality aware node identifiers to the Chord nodes. Like the original Chord protocol, it sticks to the basic primitive: given a key, it determines the node responsible for storing the key's value and it does it efficiently.

In a steady state N-node network, each node resolves all lookups via O (log N) messages. The advantage it offers is ensuring that physically close nodes are placed as close to each other in the logical ring to improve routing efficiency. We do this at the expense of increased node join time but as it is a one-time operation and node joins are generally infrequent compared to lookup operations, it is worth sacrificing. Our experiments also show that it scales well with an increase in the number of nodes, with a high lookup accuracy.

7 REFERENCES

[1] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H., 2001. Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Computer Communication Review, 31(4), pp.149-160.

[2] Rowstron, A. and Druschel, P., 2001, November. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (pp. 329-350). Springer Berlin Heidelberg

[3] Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D. and Kubiatowicz, J.D., 2004. Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on selected areas in communications, 22(1), pp.41-53

[4] H.Y.Yan, Y.LJiang, X.M.Zhou. "A Bidirectional Chord System Based on Base-k Finger Table," International Symposium on Computer Science and Computational Technology, Shanghai, China, 2008, YoU, pp. 384-388, Dec. 20-22

[5] Zheng, Xianghan, and Vladimir Oleshchuk. "Improvement of Chord overlay for P2PSIP-based communication systems." (2009).

[6] Chen, Dong, et al. "An improvement to the Chord-based P2P routing algorithm." Semantics, knowledge and grid, 2009. SKG 2009. Fifth international conference on. IEEE, 2009.

[7] Junjie, Jiang, et al. "Bi-Chord: An IMproved Approach for Lookup Routing in Chord." Lect Notes Comput Sci (2005).

[8] Cheng, Chunling, Yu Xu, and Xiaolong Xu. "Advanced chord routing algorithm based on redundant information replaced and objective resource table." Computer science and information technology (ICCSIT), 2010 3rd IEEE international conference on. Vol. 6. IEEE, 2010

[9] Zhang, Shidong, et al. "AF-Chord: An improved Chord model based adjusted finger table." Broadband Network and Multimedia Technology (IC-BNMT), 2011 4th IEEE International Conference on. IEEE, 2011

[10] Ding, Shunli, and Xiuhong Zhao. "Analysis and improvement on Chord protocol for structured P2P." Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on. IEEE, 2011

[11] Ratnasamy, Sylvia, et al. A scalable content-addressable network. Vol. 31. No. 4. ACM, 2001.

[12] Winter, Rolf, Thomas Zahn, and Jochen Schiller. "Topology-aware overlay

construction in dynamic networks." Proc. IEEE ICN'04 (2004).
[13] Feng, H., Minglu, L., Minyou, W. and Jiadi, Y.U., 2006. PChord: improvement on Chord to achieve better routing efficiency by exploiting proximity. IEICE transactions on information and systems, 89(2), pp.546-554.
[14] Xiong, J., Zhang, Y., Hong, P. and Li, J., 2005, October. Chord6: IPv6 based topology-aware Chord. In Autonomic and Autonomous Systems and International Conference on Networking and Services, 2005. ICAS-ICNS 2005. Joint International Conference on (pp. 4-4). IEEE
[15] VCL, https://vcl.ncsu.edu
[16] Averitt, Sam, et al. "Virtual computing laboratory (VCL)." Proceedings of the International Conference on the Virtual Computing Initiative. 2007
[17] Fantar, Sonia Gaied, and Habib Youssef. "Improving chord network performance using geographic coordinates." International conference on GeoSensor Networks. Springer Berlin Heidelberg, 2009
[18] Wu, Weiyu, et al. "LDHT: Locality-aware distributed hash tables." Information Networking, 2008. ICOIN 2008. International Conference on. IEEE, 2008.
[19] Taheri, Javad, and Mohammad Kazem Akbari. "TAC: A Topology-Aware Chord-based Peer-to-Peer Network." 34-42.
[20] Dao, Le Hai, and JongWon Kim. "AChord: topology-aware Chord in anycast-enabled networks." Hybrid Information Technology, 2006. ICHIT'06. International Conference on. Vol. 2. IEEE, 2006.

WORK DISTRIBUTION

Jeris Alan Jawahar - Node joins and voluntary departures, network locality updates, key migration, CLI
Sharmin Lalani - Routing, CLI, finger table optimization, keystore synchronization, dead node detection
Durgesh Kumar Gupta - Stabilization, automated deployment, testing framework, metric collection

Note: We tried testing on AWS, but could not complete it because of their configuration and security policies, due to which our RMI calls were not going through.

PAPER CONTRIBUTION

Jeris Alan Jawahar – Abstract, Introduction, Node joins and departures, Related work for network locality, Results, Conclusion
Sharmin Lalani - Abstract, Introduction, Routing, Stabilization, synchronization, Related work or finger table optimization, Limitations
Durgesh Kumar Gupta – Introduction, experimentation and results