CSCI 8790 Final Project Report Implementation of Chord Protocol

Vishnu Gowda Harish, Santosh Uttam Bobade, Anuja Pradeep Nagare

I. Introduction

HASH table is a data structure used to implement a structure that can map keys to values. A Distributed Hash Table provides look up service similar to hash tables, but is a class of decentralized distributed system where,`<key, value>` pairs are stored in a DHT. Any participating node can retrieve value associated with a given key.

”Chord” is one of the four original ”Distributed Hash Table” protocols, which was introduced by researchers at MIT in 2001(Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan).

The primary fundamental operation in peer to peer distributed systems is to lookup a value associated with an entity. Chord protocol caters to this need. It is scalable protocol useful for key look ups in distributed systems where node membership changes frequently. It takes a decentralized approach where each node maintains routing information of O (log N) other nodes in the system. It provides lookups in O (log N) messages.

In other words, Chord distributes keys over a network of nodes and implements a protocol for finding these keys. A key can be looked up from any given node in the system.

Chord address various problems of peer to peer systems which can be given as follows:

- *Load Balance*: Helps in load balancing as the keys are distributed across nodes and there are no overburdened nodes.
- *Decentralization*: Each node in the system is equally important. Improves robustness.
- *Scalability*: Chord is not too costly as the complexity is equal to the log of number of nodes in the system.
- *Availability*: Chord automatically modifies its internal data structures when nodes join and leave the system.
- *Flexible Naming*: No restrictions on naming.

The simplicity, correctness and efficiency of chord makes it useful for not only peer to peer systems but also for large scale distributed computing platforms, document and service discovery applications.

Figure 1 shows, an identifier circle consisting of the three nodes 0, 1, and 3. In this example, key 1 is located at node 1, key 2 at node 3, and key 6 at node 0 (m = 3)

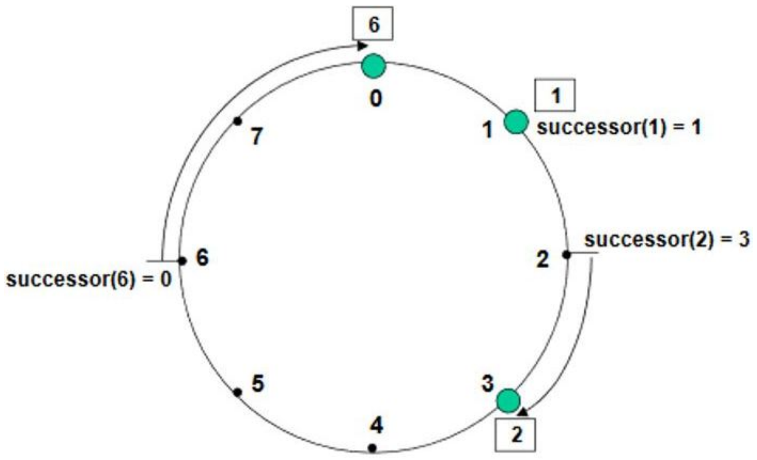


Fig. 1: An identifier circle consisting of the three nodes 0, 1, and 3. In this example, key 1 is located at node 1, key 2 at node 3, and key 6 at node 0 (m = 3)

Figure 2 shows Finger tables and key locations for a net with nodes 0, 1, and 3, and keys 1, 2, and 6

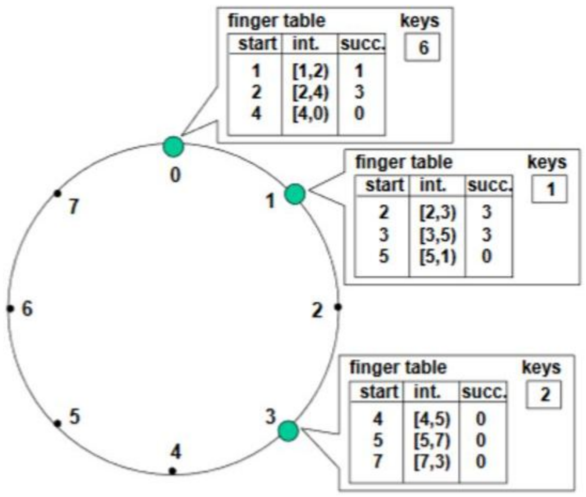


Fig. 2: Finger tables and key locations for a net with nodes 0, 1, and 3, and keys 1, 2, and 6

If a system is fault tolerant then, in Chord Decentralized system no particular node is single point of failure. Our goal is to understand how Chord protocol works.

II. Project Specifications

A. Goals

To implement a distributed system that supports the base version of Chord Protocol with the following assumptions/ features:

- The `<key, value>` pairs are immutable
- All keys are assumed to be integer type
- The nodes are considered to be immutable (cannot be removed from the network/ring)
- Dynamic Insertion of nodes is supported
- Key insertion and lookup is supported

B. Commands

Following commands will be implemented:

- get`<key>`
  Get value associated with key

- put `<key, value>`
  Put `<key, value>` into network of nodes
- node`<ID>`
  It will show a list with node ID and ID of its successor and predecessor nodes

C. Stretch goals

- System will support a graceful node exit (i.e. a node can exit the ring and it informs its predecessor and successor node when it exits the ring)
- Fault tolerance

D. Technology

- JAVA
- Socket Programming

III. Project Implementation

A. Initial State

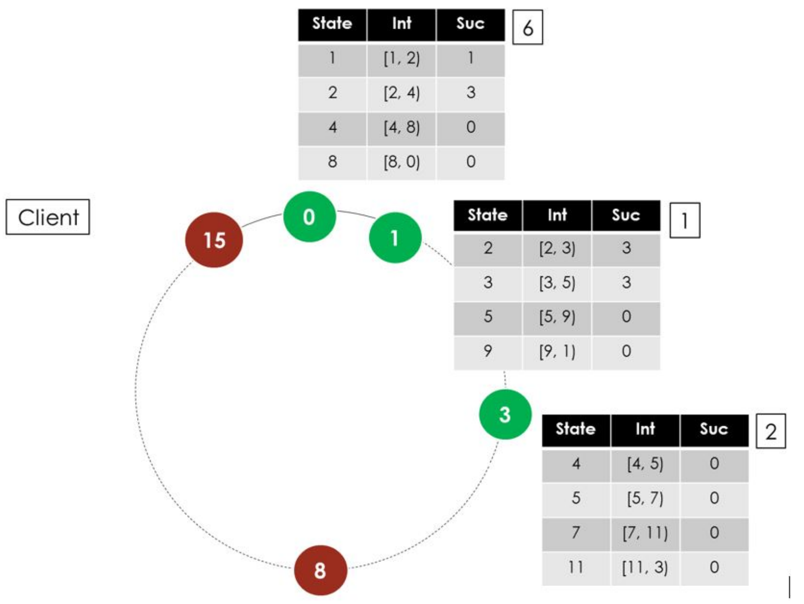
- n = 3 `(Number of Nodes)`
- m = 4 `(bits identifier)`
- Number of Keys : 3

Each node maintains a routing table which is called ”Finger Table”, it has atmost 4 entries in implemented setup, as we have 4 bits identifier. Finger Table information will be hard coded different.y for every node and is stored as a Hash Table.
Client to node request = 1 message
Node to node communication = 1 message
Node to client communication = 1 message
Figure 3 shows Finger tables and key locations for a net with nodes 0, 1, and 3, and keys 1, 2, and 6.

See Finger Table for node 0, each entry in the table is a tuple of 3 fields where, the 1st column gives information about the state, 2nd column is the Interval and the 3rd column gives the successor for the state. Node s is the ith finger of node n. First finger of n is its Immediate successor on the circle, 1st finger is referred to as successor.

$$s = succesor(n + 2^(i - 1)) \tag{1}$$

$$\begin{aligned} s &= succesor(1) && here : i = 1, n = 0 \\ &= sucessor(2) && here : i = 2, n = 0 \\ &= sucessor(4) && here : i = 3, n = 0 \\ &= sucessor(8) && here : i = 4, n = 0 \end{aligned}$$



State	Int	Suc
1	[1, 2)	1
2	[2, 4)	3
4	[4, 8)	0
8	[8, 0)	0

State	Int	Suc
2	[2, 3)	3
3	[3, 5)	3
5	[5, 9)	0
9	[9, 1)	0

State	Int	Suc
4	[4, 5)	0
5	[5, 7)	0
7	[7, 11)	0
11	[11, 3)	0

Fig. 3: Finger tables and key locations for a net with nodes 0, 1, and 3, and keys 1, 2, and 6

Node 0 points to the successor nodes of identifier

$$(n + 2^{(}i - 1))mod(2^m) \tag{2}$$

$$\begin{aligned}(n + 2^{(}i - 1))mod(2^m) &= 1 \qquad here : i = 1, n = 0\\ &= 2 \qquad here : i = 2, n = 0\\ &= 4 \qquad here : i = 3, n = 0\\ &= 8 \qquad here : i = 4, n = 0\end{aligned}$$

The successor of identifier 1 is node 1, it is the first node that follows 1, the successor of identifier 2 is node 3, successor of identifier 4 is 0 and the successor of 8 is node 0.
Node 0 wants to find successor of identifier 1. Identifier 1 belongs to interval [1,2), it belongs to 0.finger[1] node 1. Therefore, check 1st entry in finger table which shows that successor is 1.

B. Code Structure

- *NodeStarter.java :*
  This java class activates a node by creating 2 socket connections with 2 different threads. Once the class is launched both socket connections are created on the port numbers.
  Socket I: On this port, the node will wait to accept a client node connection and it will be used by client to issue commands and/or receive responses to the issued commands.
  Socket II: On this port, the node will connect to its predecessor node in the ring. Any of the requests which can not be processed by this node will be forwarded to the next node using this socket connection.

- *FingerTableEntry.java :*
  This represents a bean class for each finger table entry. As of version 1.0, it has details

such as, starting range of the finger table entry, finishing range of the finger table entry, Node ID and IP address of the successor node and the port number on which the successor will be listening on, etc.

- *SharedObject.java :*
  This class acts as a Java bean class which holds information about the finger table. This finger table contains the entries of the type `<Integer,FingerTableEntry>` and will be shared among both the threads (i.e. sockets).

- *NodeThread1.java :*
  This class is reponsible for client-node communication. It receives requests from the client in the form of:
  `[get<KeyToGet>_<msgCounter>], [put<keyToPut>_<msgCounter>]`
  It checks its own finger table to see if it has the requested key or if it's the node, which will store the new key and responds back success to the client by incrementing the msgCounter. In case if it's not the node which will store the new key or it does not have the requested key then, it will iterate over its finger table entries and find the appropriate node to process the request and forwards the request by incrementing the msgCounter.

- *NodeThread2.java :*
  It is responsible for node-node communication. It receives requests from a node in the ring. This source code works as the thread NodeThread1 to process the requests received by it and responds directly back to the client if it is a success or failure. If it's not the last node in the ring which received the current request from the other nodes in the ring, then it decides to forward the request to it's predecessor.

C. Evaluation

- To evaluate get command, lookup is performed for every key from every node, then the number of messages exchanged between each processes/nodes is checked
- Inorder to evaluate put command keys are inserted
- System is evaluated based on the number of messages exchanged
- This chord model does not support node addition

Table I shows key addition. Total keys present in system after inserting the following keys are as follows 1, 2, 6, 3, 4, 15, 12, 8, 13, 11, 10

TABLE I: Key Additions

Key	**Insert Request on Node**	**Key Inserted to Node**	**No of messages**
15	0	0	2
12	1	0	3
4	3	0	3
3	3	3	2
8	1	0	3
13	1	0	3
11	0	0	2
10	3	0	3

Table II shows the Key Lookup performed to evaluate get command for given sample keys.

TABLE II: Key Lookups

Key	**Lookup Request on Node**	**Key Present in System**	**No of Messages**
1	1	Yes	2
1	0	Yes	3
1	3	Yes	4
2	1	Yes	3
2	0	Yes	3
2	3	Yes	2
28	0	No	2
5	3	No	3

IV. Conclusion

We succesfully implemented and evaluated get and put commands as per the chord protocol. We could not complete node addition as we ran out of time. Source code of the project is submitted along with the report.

References

[1] ”http://cs.brown.edu/courses/csci1380/s15/content/projects/chord.pdf”
[2] ”https://en.wikipedia.org/wiki/Chord_(peer-to-peer)”
[3] ”https://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf”
[4] ”https://en.wikipedia.org/wiki/Hash_table”
[5] ”https://en.wikipedia.org/wiki/Distributed_hash_table”