Distributed Systems

Chord Protocol

Team Members:

Bhavin Kotak | 2018201071
Jatin Paliwal | 2018202006
Vishal Bidawatka | 2018201004

Git URL: https://github.com/bhavinkotak07/chord_protocol
Video link : https://youtu.be/2rNEjQCpFCo

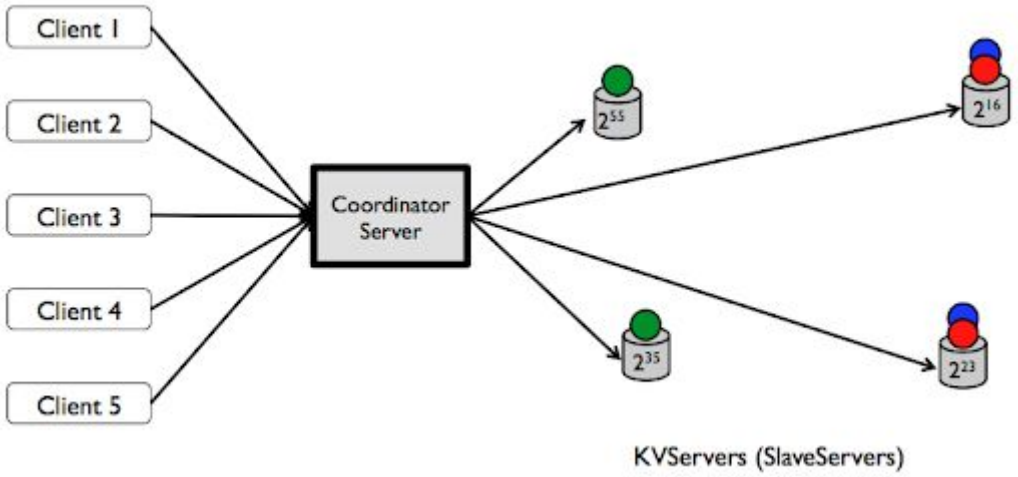
Introduction

Chord is a peer to peer distributed hash table(DHT).

1. A hash table is a data structure where you can do insert, delete and lookup in O(1).
2. DHT is a similar data structure that runs in a distributed setting.
3. In most algorithms different parts of the DHT are stored in different Nodes(processes).

Why Chord ?

Before chord we had client-server architecture based DHT.
In this kind of architecture, all the requests are routed through a single server known as Coordination server.



Disadvantages:

1. Not all nodes share same responsibility
2. Single point of failure

Advantages of using Chord:

1. Chord is a peer to peer distributed hash table that uses consistent hashing on peer's address.
2. Peers replace Slaves.
3. No single point of failure as it is fully distributed.

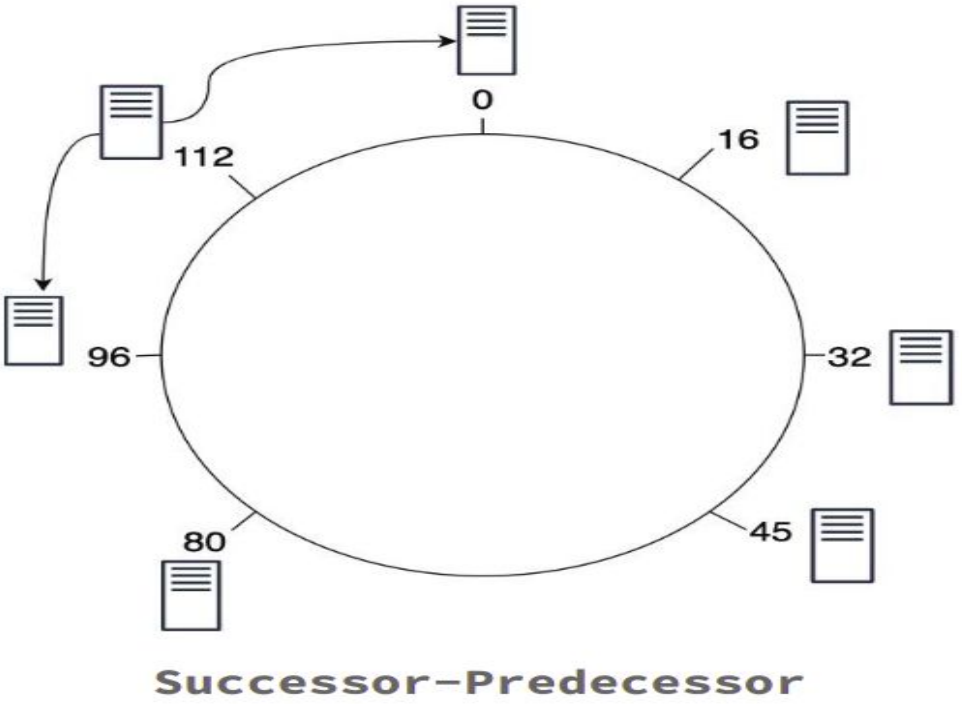
Chord Protocol

Chord uses consistent hashing to hash the peers to a particular id on the logical ring of size 2^m.

1. We could use any uniform hash function for better load balancing.
2. Assume conflict is very unlikely as N << 2^m where N is the number of peers.

There are two types of peer pointers

1. Successor - Predecessor
2. Finger table



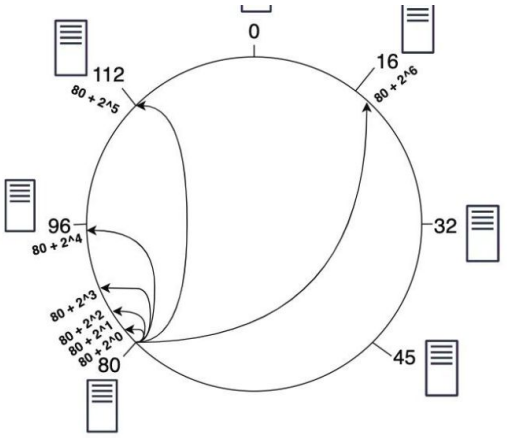
Finger table

To avoid the linear search, Chord implements a faster search method by requiring each node to keep a finger table containing up to m entries, recall that m is the number of bits in the hash key. The i^{th} entry of node n will contain a successor($(n + 2^{i-1})$ mod m). The first entry of the finger table is actually the node's immediate successor (and therefore an extra successor field is not needed). Every time a node wants to look up a key k, it will pass the query to the closest successor or predecessor (depending on the finger table) of k in its finger table (the "largest" one on the circle whose ID is smaller than k), until a node finds out the key is stored in its immediate successor.

Finger Table

Finger table for node 80

i	ft(i)
0	96
1	96
2	96
3	96
4	96
5	112
6	16



ith entry at peer with id n is first peer with id >= n + (2^i)(mod 2^m)

The above two figures show the arrangement of successor-predecessor and finger table values.

Node join

Chord needs to deal with nodes joining the system concurrently and with nodes that fail or leave voluntarily.
A basic **stabilization** protocol is used to keep nodes' successor pointers up to date, which is sufficient to guarantee the correctness of lookups. Those successor pointers are then used to verify and correct finger table entries, which allows these lookups to be fast as well as correct.
If joining nodes have affected some region of the Chord ring, a lookup that occurs before stabilization has finished can exhibit one of three behaviours.

1. The common case is that all the finger table entries involved in the lookup are reasonably current, and the lookup finds the correct successor in **O(log N)** steps.
2. The second case is where successor pointers are correct, but the fingers are inaccurate. This yields correct lookups, but they may be slower.
3. In the final case, the nodes in the affected region have incorrect successor pointers, or keys may not yet have migrated to newly joined nodes, and the lookup may fail.

Stabilization scheme guarantees to add nodes to a Chord ring in a way that preserves reachability of existing nodes, even in the face of concurrent joins and lost and reordered messages

Routing

Routing protocol for operations like insert, delete and search.

- Hash the object to get a key k in range 0 to 2^m.
- If(k is not in between predecessor(n) and n)
    - At node n, send query for key k to largest successor/finger entry <= k.
    - If none exist, send a query to the successor(n).
- Else
    - Process the query.

Number of hops: O(log(N)) Where **N** is number of Peers.

Stabilization

To **support concurrent joins**, we have implemented a stabilization module. It ensures correct lookups, all successor pointers must be up to date. Therefore, a stabilization protocol is running periodically in the background which updates finger tables and successor pointers.

The stabilization protocol works as follows:

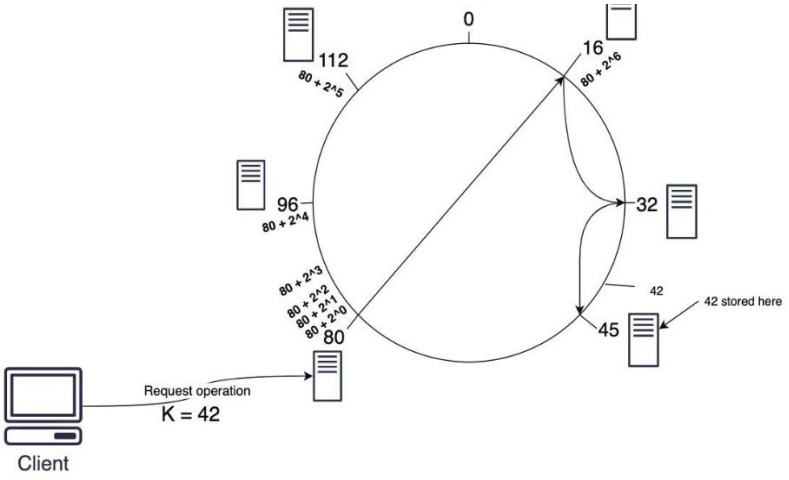
Stabilize(): Node n asks its **successor n'** to check whether it is still its **predecessor** or not, if not it asks for the **predecessor of n'** and set's **p the predecessor of n'** as its successor instead (this is the case if p recently joined the system).
Notify(): notifies n's successor of its existence, so it can change its predecessor to n
Fix_fingers(): updates finger tables periodically based on predefined rules.

Every **10 seconds, stabilize** and **fix_fingers** functions are invoked which makes sure that the finger table, successor and predecessor of all the nodes are pointing to the correct nodes.

Example

Finger table for node 80

i	ft(i)
0	96
1	96
2	96
3	96
4	96
5	112
6	16



Code repository

Git URL : https://github.com/bhavinkotak07/chord_protocol
Video link : https://youtu.be/2rNEjQCpFCo