Chord Implementation Design

Sahil Jain - 2016BSY7510,Vamsi - 2015MCS2358

February 2017

Chord Implementation Design

1 Introduction to Chord

Chord Protocol is a P2P DHT network which provides fast distributed computation of a hash function mapping keys to nodes related to them. Chord uses consistent hashing to provide keys to nodes.

Whenever a new node enters the system the keys are evenly distributed to all the nodes thereby maintaining a well distributed load. Since a chord node stores information about some of the other nodes located close to it, So chord protocol is scalable. All this information is stored in a distributed manner, so each node receives the hash value from other nodes. Every node maintains a finger table so it tends to achieve a lookup operation in O(log N) operations.

2 Chord Implementation

2.1 Global Info

A list and a sorted tree map is maintained for all nodes of the overlay network. It is used to dump information of all nodes and their finger tables for verification , inspection and debugging.

2.2 Node Implementation

Each node is implemented as a thread which keeps running in the background to perform message look ups and keep stabilizing itself after certain intervals of time.

2.2.1 Chord Node Structure

Every node has the following data structures maintained by it :
1.long threadId
2.String nodeId - String identifier for the node

3.private Node predecessor - previous node onn the chord ring
4.private Node successor - immediate node on the chord ring network
5.private DHTKey nodeKey - HashKey structure of the nodeId
6.private FingerTable fingerTable - An array of m fingers where m is the key size in bits
7.private fileList- List of files kept by each node
8.private MessageQueue¡Message¿

2.2.2 Node Functions

1.addFile - add File structure to the list
2.deleteNode - deltes itself from the global list and notifies others
3.findSuccessor - finds the successor for the DHTKey
4.stabilize - fixes its successors and predecessor
5.initFingers - initilaises fingers of the finger table
6.fixFingers - refreshes fingers periodically
7.notifySuccessor - notifies the succesor about itself
8.notifyPredecessor - notifies predecessor about itself

2.3 Node Message Structure

2.3.1 Message Types

1.FIND-SUCCESSOR
2.PUT-SUCCESSOR
3.GET-PREDECESSOR
4.PUT-PREDECESSOR
5.NOTIFY-NEW-PREDECESSOR
6.NOTIFY-NEW-SUCCESSOR
7.STABILIZE
8.EMPTY

2.3.2 Message Fields

a.public Node srcNode; // Originator of the message
b.public Node sender; // the one who writes message in someone else queue
c.public Node destNode; // Node which will be receiving it
d.public msgEnum msg-enum; // message descriptor
e.public Object data;// actual message (can be a node or a key)

3 Chord Operations

3.1 Node addition

Step1: Create a new ID for the node.
Step2: Call the Node create function which hashes the nodeID and adds it to the global data structure.
Step3: Join the network with a fixed node. In our case its the node[0] of the list always for every node.
Step4 : Find the node's successor and initialise its finger table.
Step5: Notify the new successor and successor's old predecessor about its arrival.
Step6: Stabilise the network

3.2 Node deletion

Step1: Delete the node from the global list and sorted tree map.
Step2: Notify the successor and predecessor about its departure.
Step3: Transfer keys to the successor.

3.3 File addition

Step1: Compute the hash for the file string taken as input.
Step2: Find the successor for the hashed fileName.
Step3: Add the File (name and key) to the fileList maintained by the node.

3.4 File Lookup

Step1: Compute the hash for the file string taken as input.
Step2: Find the successor for the hashed fileName.