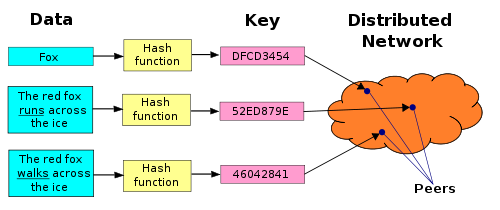
**Design Document**

In this project a simple distributed hash Map table has been implemented. The Hash Map table containing key value pair is distributed among the servers. The peers in the distributed network are connected to each other during boot up and aware of each others presence .The operations supported are put , get and delete. Each peer acts as both server and client. As client the peer decides on which server’s hashmap table the key-value pair should be stored using a Hash function.

**Principle**

The distributed hash are is a form of decentralized distributed system. This kind of system provides an efficient method for key-value retrieval and can be used to store metadata in large networks. Since the HashMap is distributed simultaneous read and write operations into the distributed hash tables on the peers in the system



Courtesy Wikipedia.com

**Advantages of a Distributed Hash Map**

Decentralized :Failure in one node does not affect the other nodes

Scalable : More amount of data can be stored along with speedy retrival of data

**Architectural design**

Peers

Each Peer acts as both a server and a client. As a client it receives a request for get/put/delete operation. Key is given as input for any of these operations. The client then uses the hashfunction to determine the server where the value with the given key should be stored retrieved or deleted from.

**Implementation**

The distributed Hash table is implemented using the following classes of java

Peer.java

Utility.java

Perf.java

ServerOperate.java

Peer.java

This is the class which contains the main function . The class starts a server thread and a client thread.

The thread executing the server code opens a server socket with the port number as specified in the in parameter ‘ServerPortNumber’ in the config file. This thread creates child threads on receiving requests from other peers in the network .

The client thread initially sets up socket connection to all the other peers . The sockets connections are then stored in a socket array. When the client needs to send messages to another peer in future it retrieves the particular socket from the array and sends the message using it.

The class also has he hashCode method which overrides the inbuild hashCode method in Java. The ‘djb2’ algorithm is used to implement the hash function.

ServerOperate.java

This class implements multithreading when the peer acts as a server. Based on the input choice from the client the get/put/delete operations are done

**Implementation details:**

**Hash Function**

The hashCode is overridden with the DJB2 algorithm implementation code. The algorithm is said to have lesser collisions and greater speed.

**HashMap Structure to store key-value**

ConcurrentMap<String, LinkedHashSet<String>>

CurrentMap is used to store the key value pair to support multithreading

Bucket

String : the key is stored as a String.

LinkedHashSet<String> : Linked hashSet is used to handle collisions when the key is same and values to be entered are different. HashSet also avoids duplicate values in the key-value pair

**Operations**

**Put Operation**

Client side

The client on receiving a key , applies the hashcode method to the key and decides on which Server the value is to be put. In case the hashFunction returns a hashcode returns that is equal to its own id then the key value is stored in its local hashMap.

If the hashCode returned is not its own ID the socket connect to the server with thatID is retrieved from the socket array and then the key value pair is sent to that server.

Server Side

On receiving a request to put a key-value pair , the server stores the key value pair in its own hashMap.

**Get Operation**

Client side

The client on receiving a key, applies the hashcode method to the key and decides in which Server the value is to be searched. In case the hashFunction returns a hashcode that is equal to its own id then the value for the key is searched local hashMap.

If the hashCode returned is not its own ID the socket connect to the server with that ID is retrieved from the socket array and then the key to be searched is sent to that server and the the result of search is received at the server

Server Side

On receiving a request to get a value of a certain key , the server searches for the key in its own hashMap , if the key is found it sends the value corresponding to the key and sends a success error code else sends failure error code.

**Delete Operation**

Client side

The client on receiving a key, applies the hashcode method to the key and decides in which Server the value is to be searched and deleted from. In case the hashFunction returns a hashcode that is equal to its own id then the value for the key is searched local hashMap and incase the key –value pair present it is deleted.

If the hashCode returned is not its own ID the socket connect to the server with that ID is retrieved from the socket array and then the key to be deleted from

Server Side

On receiving a request to delete a key , the server searches for the key in its own hashMap , if the key is found it deletes it and sends a success error code else sends failure error code.

**Use of property file**

To prevent hardcoding of values within the code , Config.properties file. The file is used as a config file.

The file contains the Ip and port number of all the peers that are in the distributed network including its own Ip and port.

The file also contains the peer’s ID .

Challenges

Choosing the hash function

An optimal hashing algorithm had to be used , it has to have speedy computation as well as good distribution.

PerformanceTesting

Running 8 servers concurrently on different servers was difficult due to constraint in resources.

Duplicate values corresponding to a Key

Prevention of duplicate values of peers corresponding to a key in the HashMap , LinkHashSet is used to avoid this.

Future enhancements

* Dynamic number of servers in the networks. This allows for the peers to enter and leave the system without disrupting the services
* Replication of data the on each of the servers. In case on of the server goes down the data is still not lost and can be accessed from the duplicate server.
* Better hashing algorithm to increase speed of computation and better distribution to reduce collisions
* Better routing algorithms to reduce latency in the network