Advanced Operating Systems

CS550

Programming Assignment 2

Distributed Hash Table

Student

Nikhil Nere

A20354957

nnere@hawk.iit.edu

Table of Contents

1. Introduction

1.1. Purpose

This document provides a detailed design of the Distributed Hash Table design. It provides an overview how different components in the system interact with each other.

1.2. Problem Statement

A distributed hash table is to be implemented which has multiple indexing servers to store the key-value pair. The peers in the network can put, get and delete the key-value pair on the distributed hash table.

Servers: Each server in the system will store the key value pairs in a hash table. When a peer will request to put a key-value pair, the server will store the key-value pair in the hash table and will return true if storing was successful. When a peer will send a get request the server will return the value. For delete request the server will delete the key-value pair and will return true if deleted successfully.

Client: A client can perform put, get and delete operations on the distributed hash table. The client will first identify which server to go to using a hash function. A hash will be calculated for the given key which will uniquely identify a server in the system.

It's a two level hashing. First at the client side to identify which server to go to. Second at the sever side to store the key-value pair in the hash table.

2. Design Overview

2.1. Topology Diagram

The system has multiple servers and clients

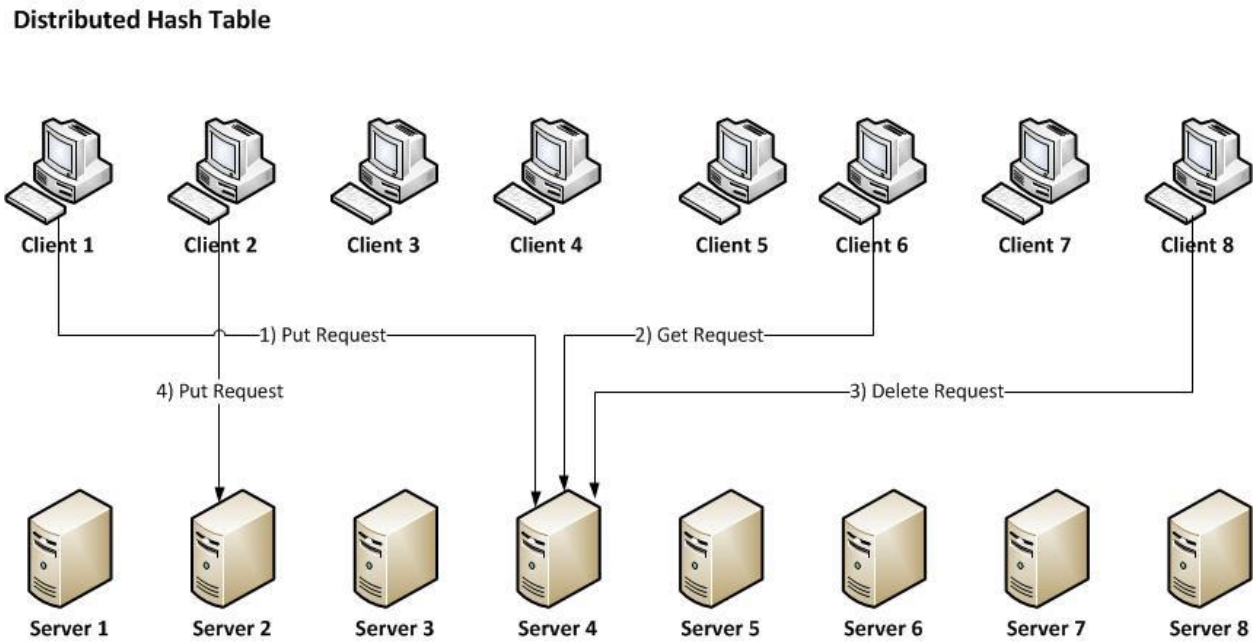

Figure-1: Distributed Hash Table

2.2. Detailed Design of the System

- The system has multiple servers which will store the key-value pair in a hash table. A client in the system can store a key-value pair on the server.
- At the client side a hash function will decide on which server the key-value pair should be stored. Once the key-value pair is stored on one of the servers, it will be available for all the clients in the system.
- Any client in the system can send a get request with the key and will get the value. For a get request same hash function will used to identify which server will have the key-value pair.

Below is detailed communication between different components. Please refer to **Figure-1**.

1) Client 1 sends a put request to server 4. The decision of connecting to server 4 is taken using the hash function.

On receiving put request server 4 will store the key-value pair in a hash table and will return true. Now any client can request for the key value pair.

2) Client 6 sends a get request with the key that was stored by client 1. Again the hash function will decide to which server to go to.
   Server 4 will receive the get request and will return the associated value from the hash table.
3) Client 8 sends a delete request with the same key stored by client1. The hash function will tell to go to server 4.
   Server 4 will receive the delete request and it will delete the key-value pair from the hash table. After deletion is successful server 4 will return true.
4) Client 2 sends a put request with new key-value pair. This time the hash function decides to go to server2.
   Server2 will receive the put request and will store the key-value pair in the hash table.
   Any client in the system can now perform get and delete operations on this key-value pair.

This is how the components in the system interact with each other.

All the servers in the system are always listening for the requests from clients and they can distinguish between get, put and delete requests. Any client can store a key-value pair on the servers making it available for other clients to get and to delete.

3. Implementation

3.1. Server Component

- All the servers handle the clients independently by creating an independent thread for each client.

```
while (true){
    Socket client = indexServerSocket.accept();

    Thread hashTable = new Thread(new ServerHashTable(client));
    hashTable.start();
}
```

- The server identifies the type of request if it is a put, get or delete request

```
if ("0".equals(reqType)){
    outToClient.writeObject(put(request));
    outToClient.flush();
}else if ("1".equals(reqType)){
    outToClient.writeObject(get(request));
    outToClient.flush();
}else if ("2".equals(reqType)){
    outToClient.writeObject(delete(request));
    outToClient.flush();
}
```

- The put, get and delete methods perform the Map operations.

```
private boolean put(String request) {
    int index = request.indexOf('|');
    String key = request.substring(0, index);
    String value = request.substring(index+1, request.length());
    distHashTable.put(key, value);
    return true;
}

private String get(String key) {
    return distHashTable.get(key);
}

private boolean delete(String key) {
    distHashTable.remove(key);
    return true;
}
```

3.2. Client Component

Client component creates connections with all the servers at start up and keeps the connections open for better performance.

- **Opening connections**: Connection is established with all the servers at the beginning and they are never closed.

```
for (int i = 0; i < NUM_OF_SERVERS; i++){
    try {
        sockets.add(new Socket(serverIPs.get(i), serverPorts.get(i)));
        objOutStreams.add(new ObjectOutputStream(sockets.get(i).getOutputStream()));
        objInStreams.add(new ObjectInputStream(sockets.get(i).getInputStream()));

    } catch (IOException e) {
```

- **Hash Function:** Hash function is applied to the key. The hash calculated is the server number.

```
private int getHash(String key) {
    char ch[];
    ch = key.toCharArray();
    int sum = 0;
    for (int i = 0; i < key.length(); i++){
        sum += ch[i];
    }
    return sum % NUM_OF_SERVERS;
}
```

- **Distributed hash table interface:** The client provide an interface to put, get and delete from the distributed hash table.

```
public interface DistributedHashTable {
    public boolean put (String key, String value);
    public String get (String key);
    public boolean delete (String key);
}
```

4. Future Enhancements

- Currently the system has a fix number of servers, it can be made flexible.
- It is not possible to add or remove a server from the system. The system can be made scalable.
- A sophisticated hash function can be implemented to ensure the keys are evenly distributed among all the servers.
- The system can be made persistent.

5. Assumptions and Constraints

- All the servers should be up and running before making any get, put or delete request.
- If one of the servers goes down then it will result in breakdown of the entire system.
- Before running the client the port should be free on which the server will be listening. Or make sure a free port is given in the properties file.
- If running multiple servers on a single machine then give different port numbers for each instance of the server