Design Document

Decentralized File Sharing System

Programming Assignment 3

TABLE OF CONTENTS

1. **Introduction**
    a. **Purpose**
        - It is possible for single server to hold Hash Table and to handle all requests.
        - Purpose of the simple Decentralized File Sharing System is to high performance computing, scalability, fault tolerance, persistence.
        - The main advantage of hash tables over other table data structures is speed.
        - Along with Distributed Hash Table will use the file sharing system in which each client/server register its own file. Unlike centralize file sharing system we can save registry on distributed system and we can manage replica of Hash so that we can reduce centralize dependency.
    b. **Scope**
        - This system is designed to share the Hash Table across multiple peers (Client/Server) connected over network. There are mainly 3 functions
        - **Register**- Each peer (Client/Server) can call register function and make Key-Value pair entry on any server. Server will get selected by result of hash function. Key is file name which we are registering at that time.
        - **Search**- Each peer can search for other peer's file and can download the file with the help of this function.
    c. **Acronyms, Abbreviations and Definition**
        - **Peer** – is Machine on server as client or server.
    d. **References**
        - http://datasys.cs.iit.edu/projects/ZHT/
        - https://en.wikipedia.org/wiki/Hash_table
        - To understand the Distributed System Architecture and its original intent

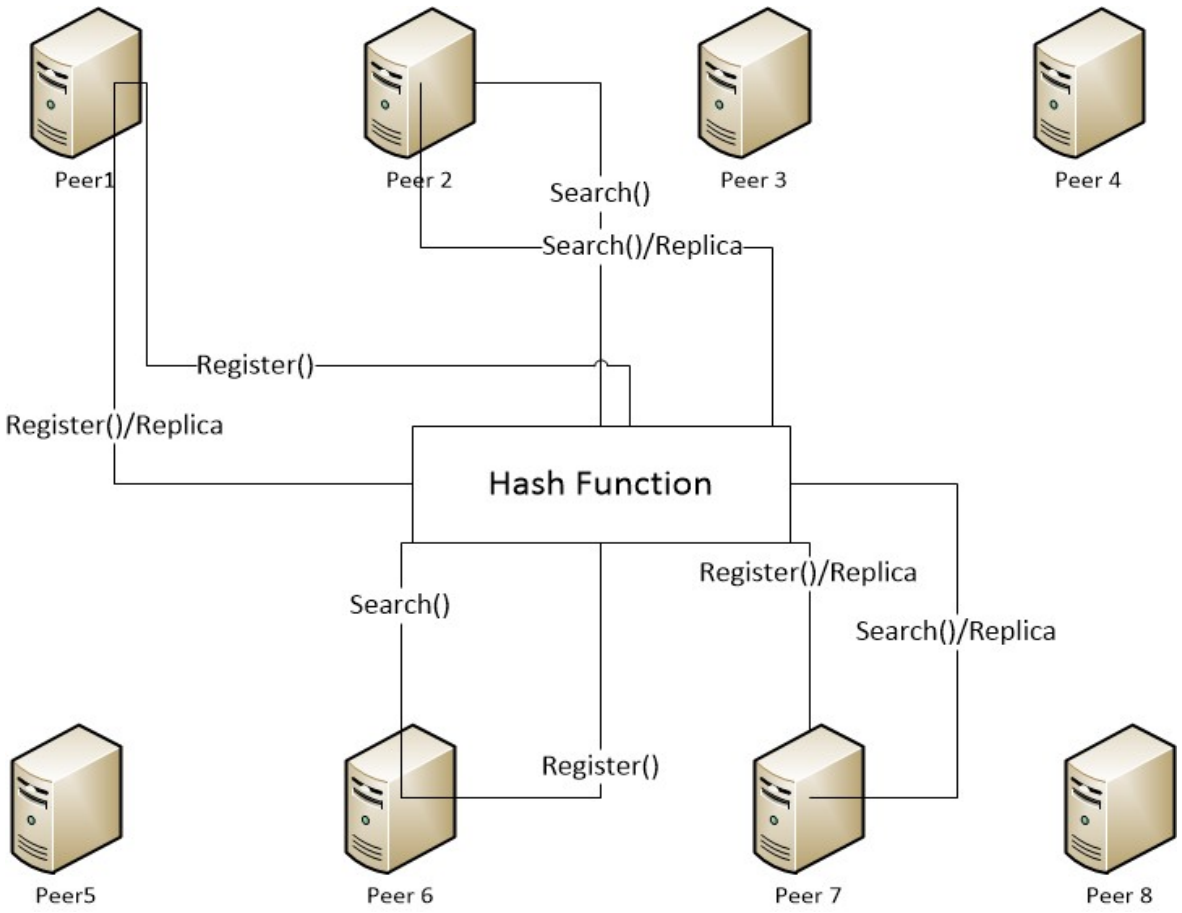
2. **System Overview**
    a. **Functionality**
    1. In Decentralized File Sharing System, peers use to communicate with each other for multiple reasons. Like file downloading.
    2. Firstly Server in our case peer (Client/Server) gets start and put itself in ready state to accept request from other peer for 3 reasons either to **Register** or to **Search** or **Download** the files.
    3. In **Register** call, peer sends a Key (file name) to the hash function and which will return the addresses of server, where that Key-Value pair (here value is the address (IP + Port) of peer who called the register function) will get stored.
    4. After receiving **Register** call server stores the Key and Value in Hash Table in the format of that is <Key, Value> pair.
    5. If same Key is present in Hash Table Server will check whether value associated with that is present in Hash Table or not if it is present it will get skipped otherwise server simply append the new value with existing value in hash table.
    6. In **Search** call, Peer will send the Key (file name) to hash function, then hash function again will decide which server has that key accordingly request will route to that server. Server will search for specific Key send by peer on successful search, Server returns the value associated with that Key otherwise unsuccessful search message to client.
    7. If peer get successful response. Server will present options to client like file is present on number of server and choose any server to get that file download. By

choosing any server client can download file successfully.

8. **Fault Tolerance –** Our system support the fault tolerance mechanism, Once all connection established successfully. We can test this by killing any number of peers. Note that before killing any peer make sure they all are connected before. (for that you need to either **register** or **search** by each peer in Distributed system)
9. **Replication,** we have one extra feature here called replication factor.
   i. Replication factor is a number which use to indicate how many copies of file replica and Hash table we have to maintain.
   ii. **Register** – If replication factor is greater than 0, in register method client will add number of replica of Hash Table and file copies on distributed network other than main copy of Hash Table.
   iii. **Search –** If replication factor is greater than 0, in search method client will search for main copy of hash table. If server holds main copy is down serach will continue with next replica of Hash Table till all copies get scanned OR get successful response.
   iv. **Download** – As we have to support data resilience to ensure data is not get lost on node failure. We kept copies of Hash Table on multiple server along with copies of file. So if any server hold file is down user can move to next available server hold the copy of that file and can download the file.

3. System Architecture

a. High Level Architecture



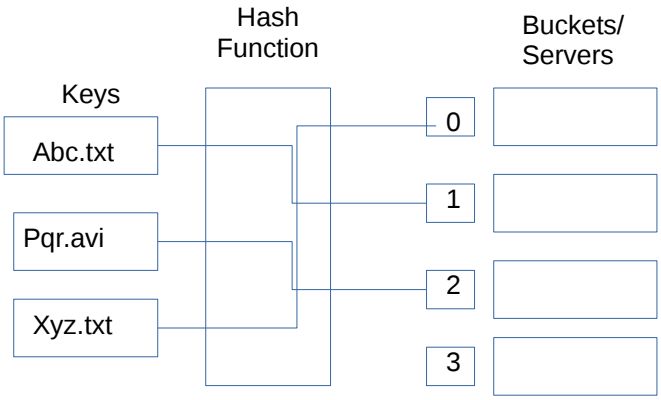
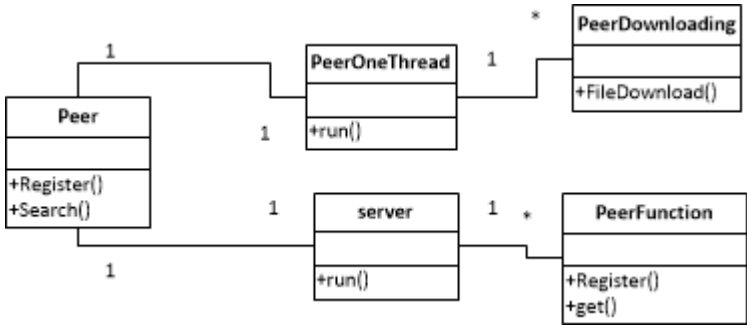


fig. Hash function working

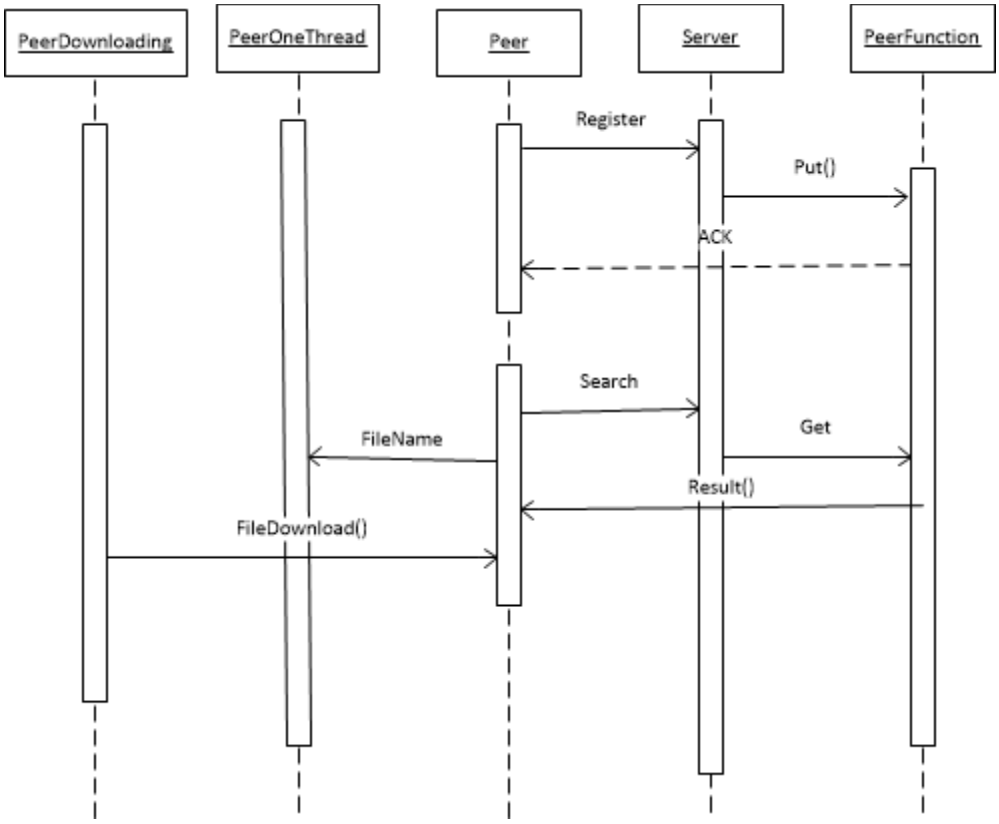
As shown in diagram Distributed Hash Table system is consisting of below components.

1. **Peer** – Requesting for storing Key-Value pair on server. Requesting value associated with Key.
2. **Hash Function** – Peer enter the Key and value to store on server. Hash function on local machine finalize the server where that Key-Value is going to be saved. Similarly hash will help in searching operation to get Server address which contain that Key-Values.
3. **Server/Bucket** – Server is system who store the value and Key associated with value in Hash Table. And if replica is on then copy of file is also get saved on server.

b. Class Diagram –



c. **Sequence Diagram –**



4. Technical Overview

a. Server
- Server is a server side class. It shall always listing for new requests from client/peer.
- As soon as clients get connected with this server it will create new thread for requesting client. And again go on listing for new client request.
- PeerOneThreads is server side class. It shall always listing for file downloading request from client/peer.

b. Peer/Client
- Peer class could be client requesting for Register, Search functionality.

5. Component Description

There are 2 main components in Distributed Hash Table System

a. Server
b. Peer

6. Software Requirement

a. **Java** installed on machine
b. **Ant** for building Java Application

7. Future Scope

a. Persistence- There should be way like file. To store/capture the current state of Hash Table and saved them for future use. If server goes down and if we make it start again it should have last working hash table details.

Note -
- This application support file size larger than 4 GB.
- This application support text as well as binary files
- This application support Data Resilience, Fault tolerance, and Replication factor
- This application is developed in JAVA with abstraction socket,threading.