DESIGN DOCUMENT

DE-CENTRALIZED P2P SYSTEM

ASSIGNMENT 3

DESIGNED BY:

SAMI AHMAD KHAN
A20352677
COURSE: CS 542 ADVANCE OPERATING SYSTEM

INTRODUCTION

This project assignment is Java Based Decentralized/ Distributed Peer to Peer(P2P) File Sharing System. Point-to-Point (P2P) technology enables the sharing of computer resources such as files by a direct exchange between end-users computers. P2P networking means files are not stored on a central server. Instead, client software (such as the popular Kazaa, Limewire) works as a server for shared files on an individual's computer. This allows each computer with the software to act as a mini-server from which other P2P users can download files.

This Distributed P2P system is implemented using the concept of **Distributed Hash table (DHT).**

A **distributed hash table** (**DHT**) is a class of a decentralized distributed system that provides a lookup service similar to a hash table: (*key*, *value*) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows a DHT to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

It is one of the fundamental algorithms used in scalable distributed systems like web caching, **P2P systems**, distributed file systems, etc.

OBJECTIVE:

The objective of this assignment is to design a Decentralized P2P application. In this assignment, each Peer should be acting as both the Server and the Client.

As a Client, it should connect to the server and update its files and the search results and as a Server, it should accept queries from other peers, check for matches against its local hash table and responds with corresponding results. In addition, since there is no central indexing server, search is done through consistent hashing.

The system implements the following functions:

- **A decentralized indexing server.** This server indexes the contents of all of the peers that register with it. It also provides search facility to peers. In our simple version,

you don't need to implement sophisticated searching algorithms; an exact match will be fine. Minimally, the server should provide the following interface to the peer clients:

registry(peer id, file name, ...) -- invoked by a peer to register all its files with the indexing server. The server then builds the index for the peer. Other sophisticated algorithms such as automatic indexing are not required, but feel free to do whatever is reasonable. You may provide optional information to the server to make it more 'real', such as the clients' bandwidth, etc.

search(file name) -- this procedure should search the index and return all the matching peers to the requestor.

- **A peer.** A peer is both a client and a server. As a client, the user specifies a file name with the indexing server using "lookup". The indexing server returns a list of all other peers that hold the file. The user can pick one such peer and the client then connects to this peer and downloads the file. As a server, the peer waits for requests from other peers and sends the requested file when receiving a request. Minimally, the peer server should provide the following interface to the peer client:

• **obtain(file name)** -- invoked by a peer to download a file from another peer.

SCOPE:

- Here in this assignment, I have kept the structure of the network and server as static (as told) i.e. there are 8 static servers and 8 static clients having files. Network and servers will be initialized using a config.properties file that is read by each server at startup time.
- This will be a de-centralized P2P network with each Server having their own hash table; collectively forming a Distributed Hash Table.
- The system has support for both **Text** as well as **Binary** files.

TOOL USED:

The tool used in designing this system is Eclipse Luna 4.4

DESIGNING AND DOCUMENTATION

OVERVIEW

Following are the steps in the designing of the program:

1. Creating a configurable file as “config.properties”, which will contain all the information about the 8 static servers like their IP address and their corresponding Port numbers.
2. Creating 8 static servers from the “config.properties” file.
3. Creating 8 static clients that will perform UPDATE, SEARCH, DOWNLOAD and DELETE operations on the servers
4. Both 8 servers and clients makes for 8 peers in actual.
5. Managing files transfer in DHT in terms of keys and value pairs in the Distributed Hash Table
6. Start the client
7. Perform UPDATE operation first
8. Updates all the file names in different server’s distributed hash table, dividing and distributing them equally.
9. Perform SEARCH operation
10. Retrieves Peers containing the required File from the DHT
11. Perform DOWNLOAD operation
12. Downloads the fie from other Client
13. Perform DELETE operation
14. Deletes a required file name from the network
15. Exit from the network
16. Display relevant message

***KEY FEATURES OF THIS PROJECT ASSIGNMENT 3**

1. **Large File Transfer** up to 1.5 GB can be done easily
2. **File Replication** added
3. **File Name Replication** added
4. **System Resilience** – Improved Fault tolerance added.

INPUT REQUIREMENTS

Server Startup: User/Server Admin need to enter a server number/ID that it wants to start. Before starting the client, first connect all the 8 servers.

Peer Login: As soon as the users run their peer, they will be needing to give IP address and port number of the server that they want to connect.

Online-clients: Once the client and server gets successfully connected, the client will be shown with a menu and will be asked to select an operation that it wants to perform like; Put, Get or Delete.

Operate: The client has to start from Put operation and then the rest.

SOFTWARE REQUIREMENT

- Windows, Linux, Macintosh
- Java Virtual machine

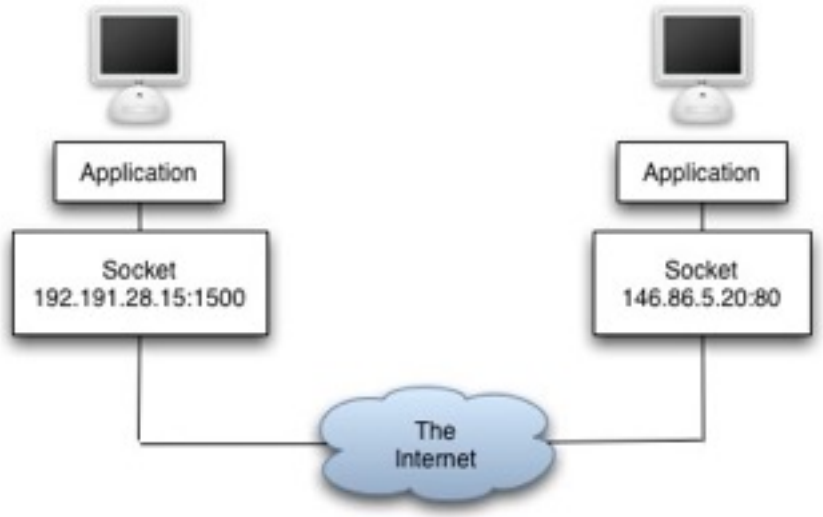
PROGRAM LOGIC AND DESCRIPTION

This P2P system is based on *Server-Client network logic* and the connection between server and client is programmed using the following programming concepts and logics:

Socket Programming

Sockets: A socket is one endpoint of a two-way communication link between two programs (in my case a server-client and peer-to-peer) running on the network. A socket is bounded by a unique port number so that the TCP layer can identify the application that data is destined to be sent to. An endpoint is a combination of an *IP address* and a *port number.*

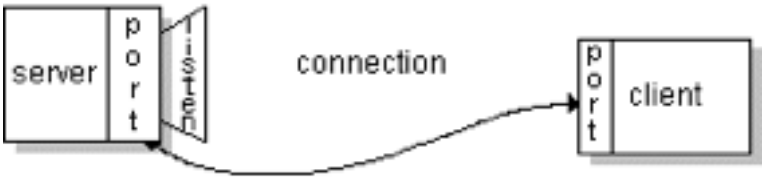
TCP provides a reliable, point-to-point communication channel to those client-server applications on the Internet to communicate with each other. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server each reads from and writes to the socket bound to the connection.



Ports: A port is a logical connection to a computer and is identified by a number in the range 1-65535. Ports are allowed to all computers attached to a network. Each port may be dedicated to a particular server/service. Port numbers in the range 1-1023 are normally set aside for the use of specified standard services. For example, port 80 is normally used by Web servers for HTTP.

In most applications, of course, there are likely to be multiple clients wanting the same service at the same time. A common example of this requirement is that of multiple browsers (quite possibly thousands of them) wanting Web pages from the same server. The server, of course, needs some way of distinguishing between clients and keeping their dialogues separate from each other. This is achieved via the use of sockets.

Client Side



Server Side

Working of a TCP Connection between Server and Peers using Sockets:

The following steps occur when establishing a TCP connection between two computers using sockets:

- The server instantiates a *ServerSocket object*, denoting which port number communication is to occur on.

- The server invokes the *accept()* method of the ServerSocket class. This method waits until a client connects to the server on the given port.

- After the server is waiting, a client instantiates a *Socket object*, specifying the *server name* and *port number* to connect to.

- The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.

- On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket. **Threads:** Threading is a facility to allow multiple activities to coexist within a single process. Threads are like lightweight processes and are independent, have concurrent path of execution through a program and each thread has it's own stacked it's own local variables. A process can support multiple threads, which appear to execute simultaneously and asynchronously to each other. And hence is very suitable to my program because using threads I was able to implement

multithreading of server and peers i.e. simultaneous communication between the server-client and peer-peer can take place.

Benefits of using threads:

• Increase the responsiveness of the application.• Take advantage of multiprocessor systems.• Simplify program logic when there are multiple independent entities. • Perform blocking I/O without blocking the entire program.

NETWORK ARCHITECTURE

Client - Server Model

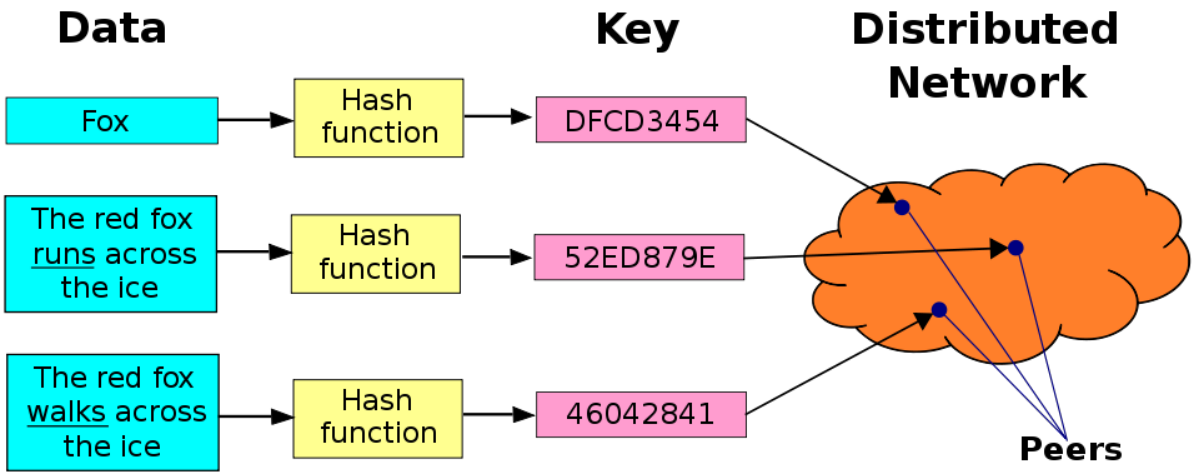
The client/server model is a computing model that acts as distributed application which partitions tasks or workloads between the providers of a resource or service, called **servers**, and service requesters, called **clients**

Distributed Hash Tables (DHT)

What is a DHT?

- Distribute data over a large P2P network
  - Quickly find any given item
  - Can also distribute responsibility for data storage

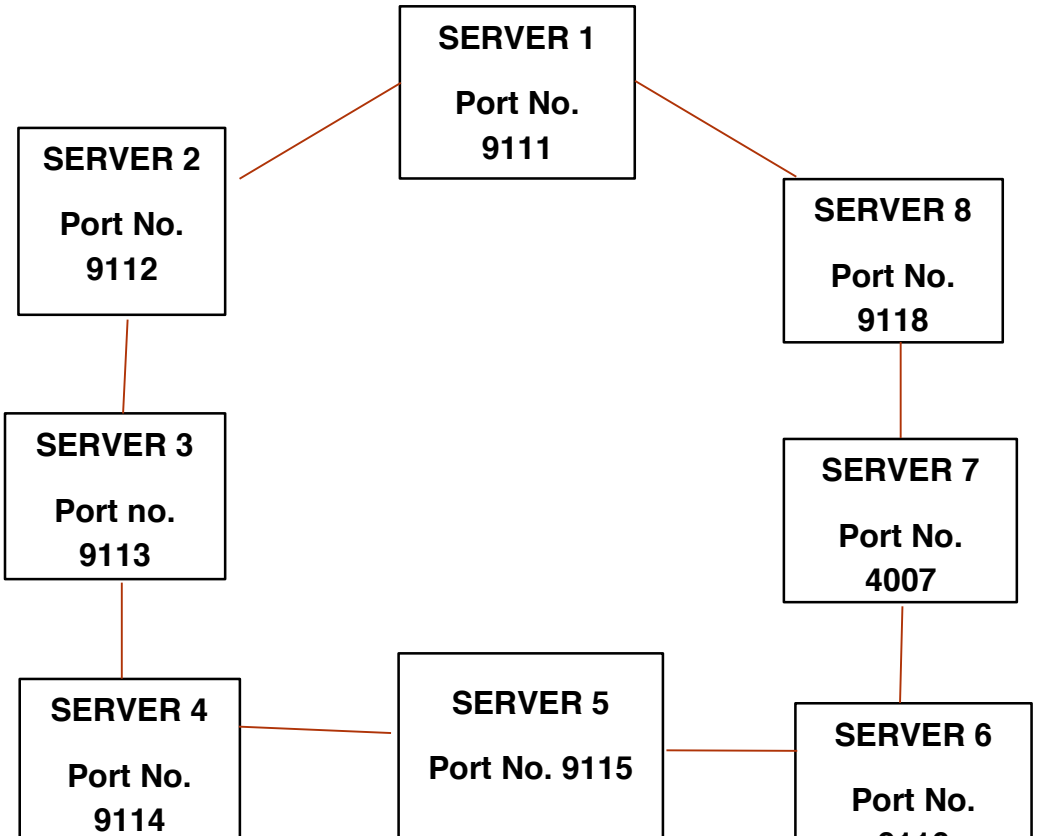
- What's stored is key/value pairs
  - The key value controls which node(s) stores the value – Each node is responsible for some section of the space
- Basic operations
  - Store(key, value)
  - Retrieve/Get(key)
  - Delete (key, value)



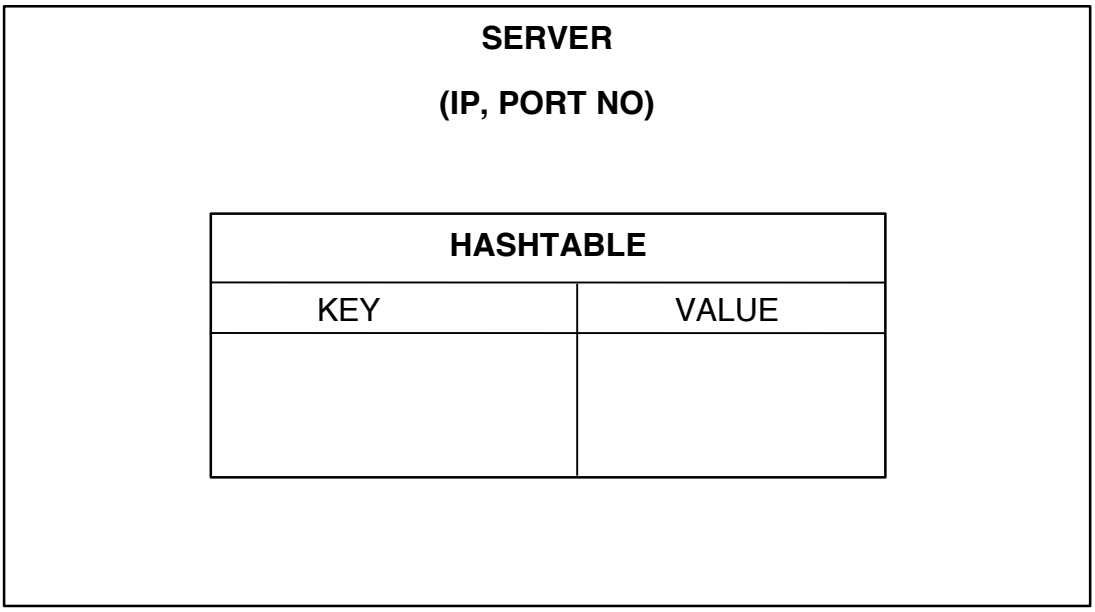
The structure of a DHT can be decomposed into several main components.[8][9] The foundation is an abstract **keyspace**, such as the set of 160-bit strings. A **keyspace partitioning** scheme splits ownership of this keyspace among the participating nodes. An **overlay network** then connects the nodes, allowing them to find the owner of any given key in the keyspace.
Once these components are in place, a typical use of the DHT for storage and retrieval might proceed as follows. Suppose the keyspace is the set of 160-bit strings. To store a file with given filename and data in the DHT, the SHA-1 hash of filename is generated, producing a 160-bit key, and a message **put(key, value)** is sent to any node participating in the DHT. The message is forwarded from node to node through the overlay network until it reaches the single node responsible for key k as specified by the keyspace partitioning. That node then stores the key and the data. Any other client can then retrieve the contents of the file by again hashing filename to produce k and asking any DHT node to find the data associated with k with a message get(key). The message will again be routed through the overlay to the node responsible for k, which will reply with the stored data.
The keyspace partitioning and overlay network components are described below with the goal of capturing the principal ideas common to most DHTs; many designs differ in the details.

DESIGN MODEL AND EXPLANATION

1. First connection is made to all the 8 servers, that are read from the "config.properties" file as shown below

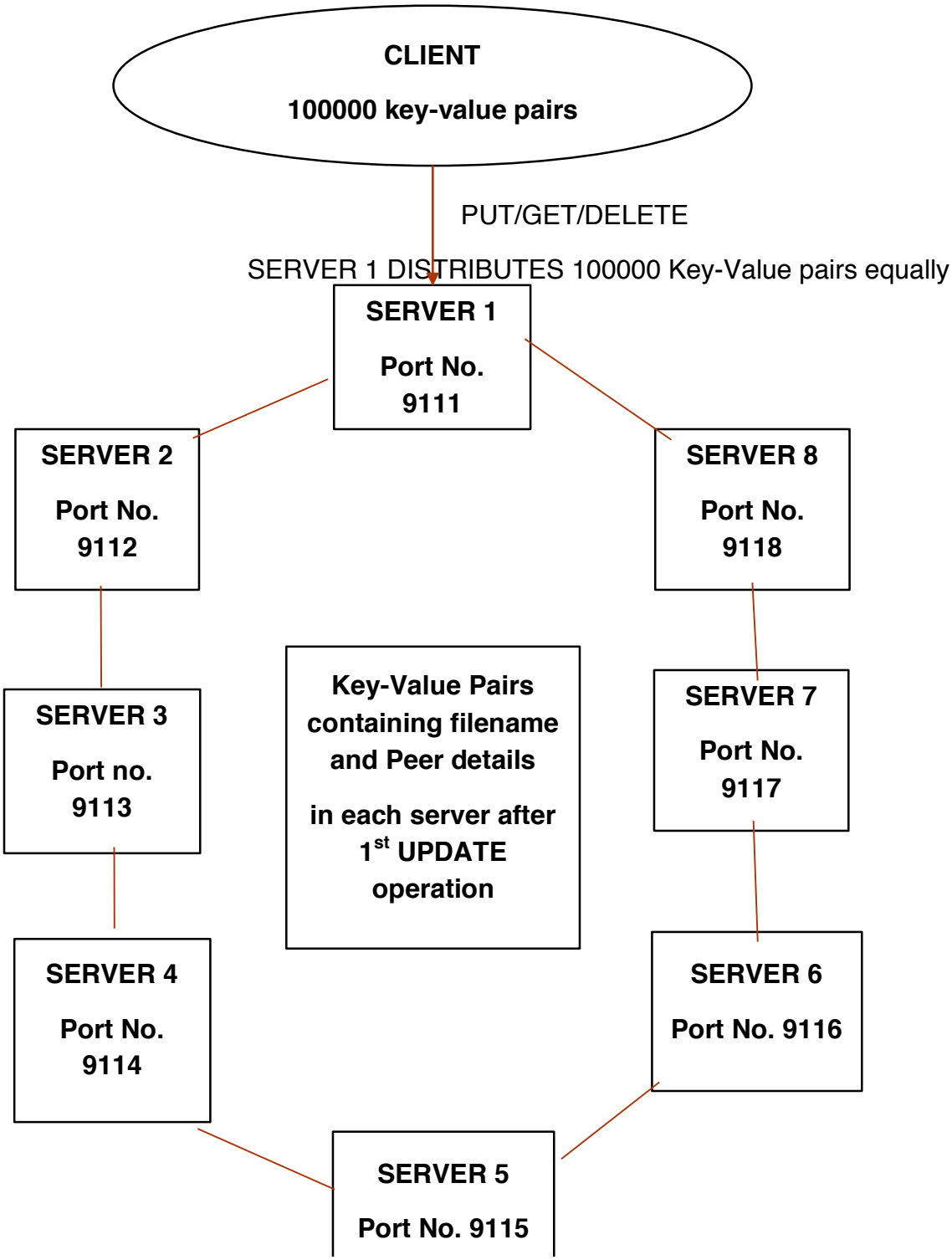


2. Each Server creates it's own private Hash Table as shown below;



Like this all the 8 servers will be having their own Hashtable where they will be storing

3. When Clients arrive they first stores the Keys and their respective value in these tables in a distributed manner using PUT operation. For example when client puts 100000 key-value pairs in the first server that it is connected to. That server will divide those key-value pairs equally and store it in each server's local hash table. And hence collectively they form a Distributed Hash Table (DHT).



LIMITATIONS OF THE PROGRAM

Following are the limitations of this program:

1. The number of servers and clients in this Distributed P2P system is static
2. Lack of GUI
3. No ring methodology used that could have improved the performance of the system.
4. The config file has to be stored in the Java Application's root folder for the program to read from the file.

IMPROVEMENTS THAT CAN BE DONE TO MAKE IT BETTER

1. Making server and clients entry into the system dynamic and not static
2. GUI implementation
3. Use of Chords or pastry could have made the system better
4. File transfer can be made using the same topology.