Design Document :

1. Details About file structure :

Files present in my code are as follows:

i. Client.java
ii. PeerServer_Intance.java
iii. PeerServer.java
iv. PeerUploadServer_Intance.java
v. PeerUploadServer.java
vi. SocketConstants.java
vii. config.property

Details about files :

I. Client.java :

* As per requirement of assignment 3, each peer should act as Index server. And at the same time, each peer should provide file uploading facility to other peers.
* So each peer will have two servers. First one is 'indexing server' and another is 'file uploading server'.
* 'Client.java' is main file of program. This file only takes care to start all server and handle all user's interaction.
* Operations OPTAIN, REGISTER, SEARCH are handle by this file.

II. PeerServer.java & PeerServer_Intance.java :

* 'PeerServer' is the indexing server files.
* It creates thread and provide non-blocking service to the peer who want request for data to indecing server.
* 'PeerServer.java' will create thread and calls 'PeerServer_Instance' file.

III. PeerUploadServer & PeerUpload_Instance.java :

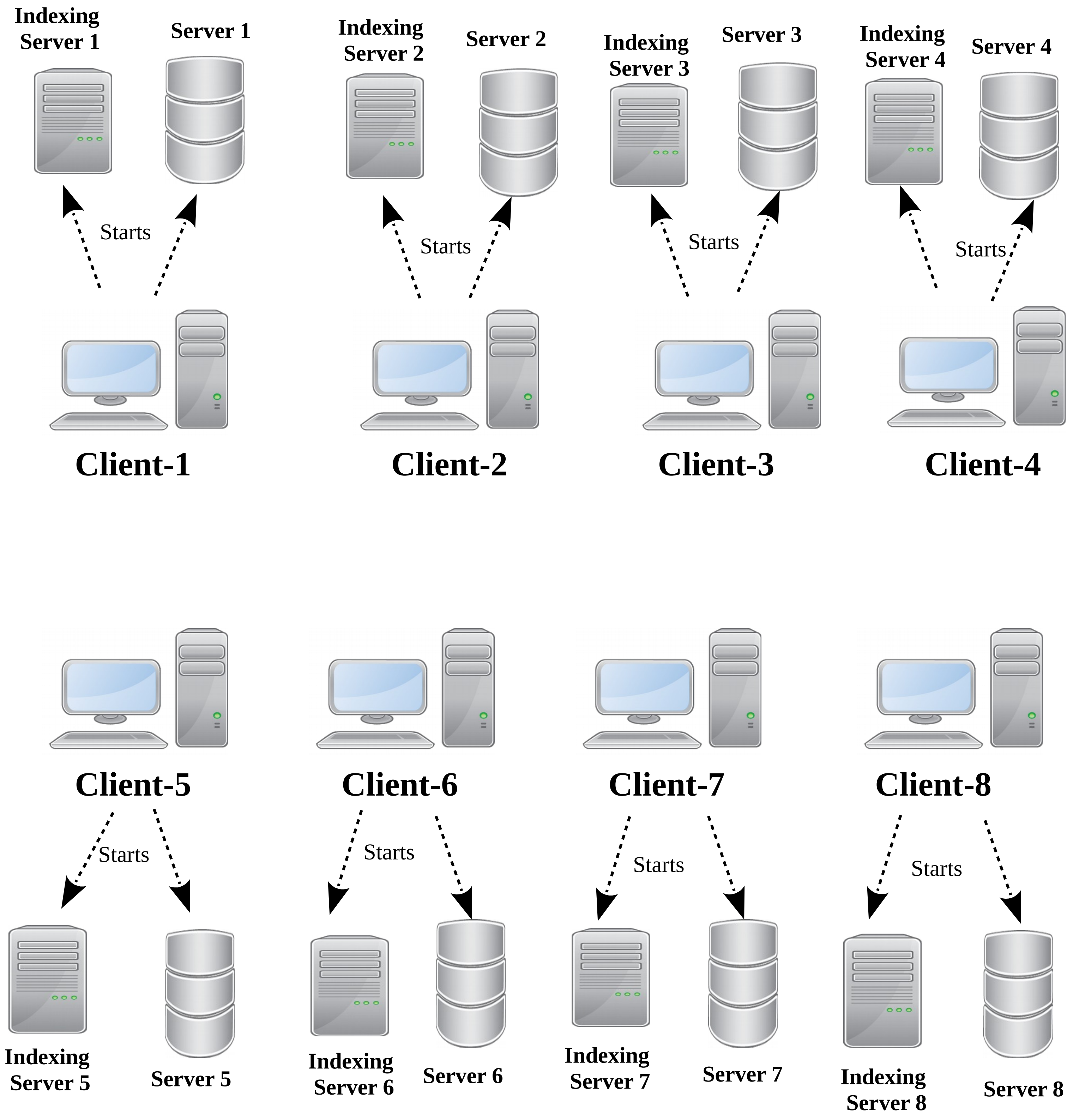
* To provide non-blocking upload service to each peer, PeerUploadServer will create thread instance called PeerUpload_Instance.

IV. Config.preperty :

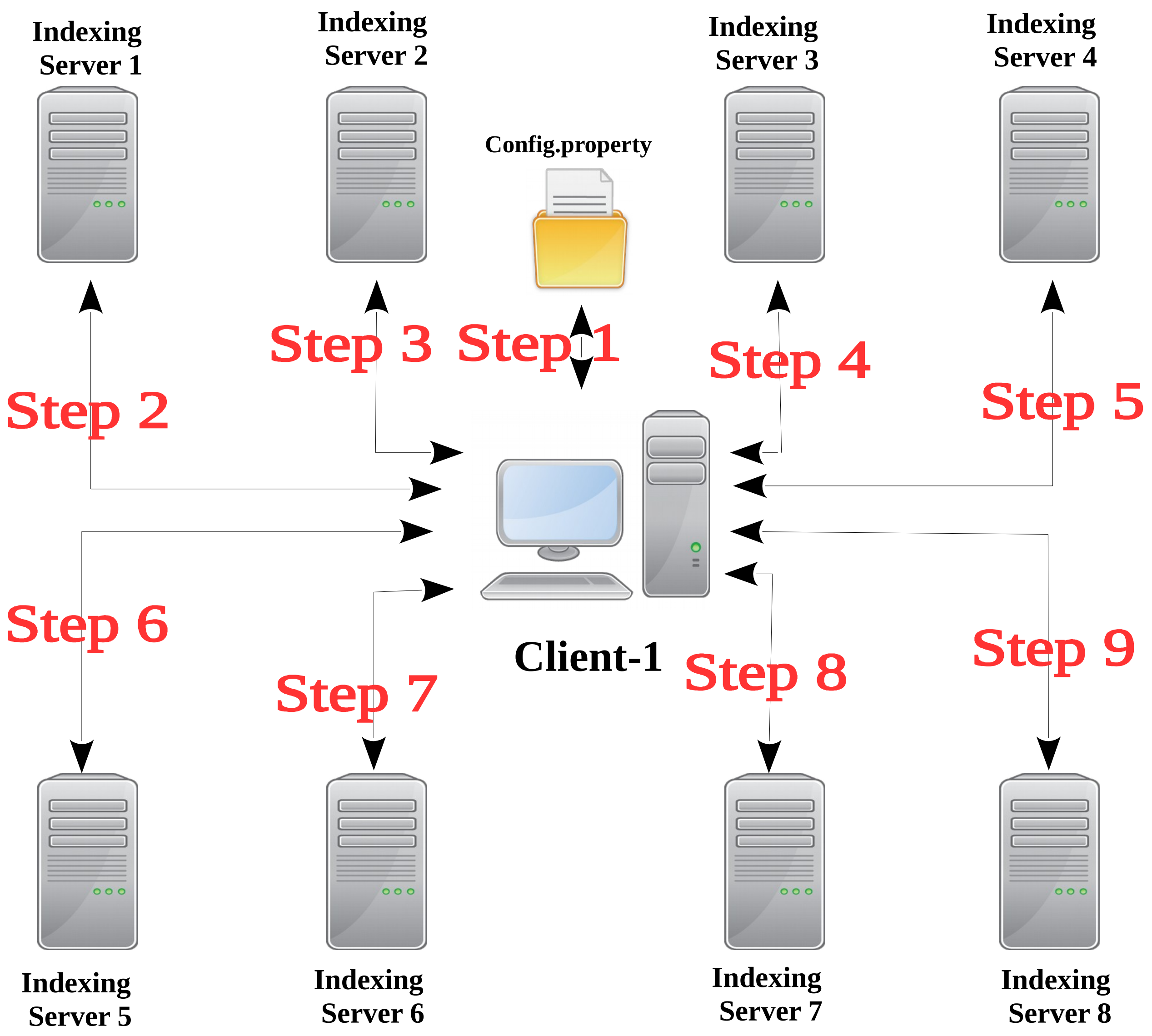
* This file will contain all constant parameter required for a program. Such as Indexing server port number and Uploading server port number.
* It also contains IP and Port address of all other Peers where distributed Indexing server resides.

I have tried to explain all the design document in visual format as follows:

2. At Start, all clients (Peers) starts two servers. First is 'Indexing Server' and second is 'Server'. Peer will take port values from '**config.property**' file. “UploadServerPort” value is for 'Server' and “ServerPort” value for 'Indexing server'.



3. After that each client fetch information from 'config.property' file. If it contains TotalPeer = 8, then it will try to connect to all 8 clients (including himself). Following figure shows, how client-1 establishes connection with all other Client's server. Similarly all other clients try to connect with each other.



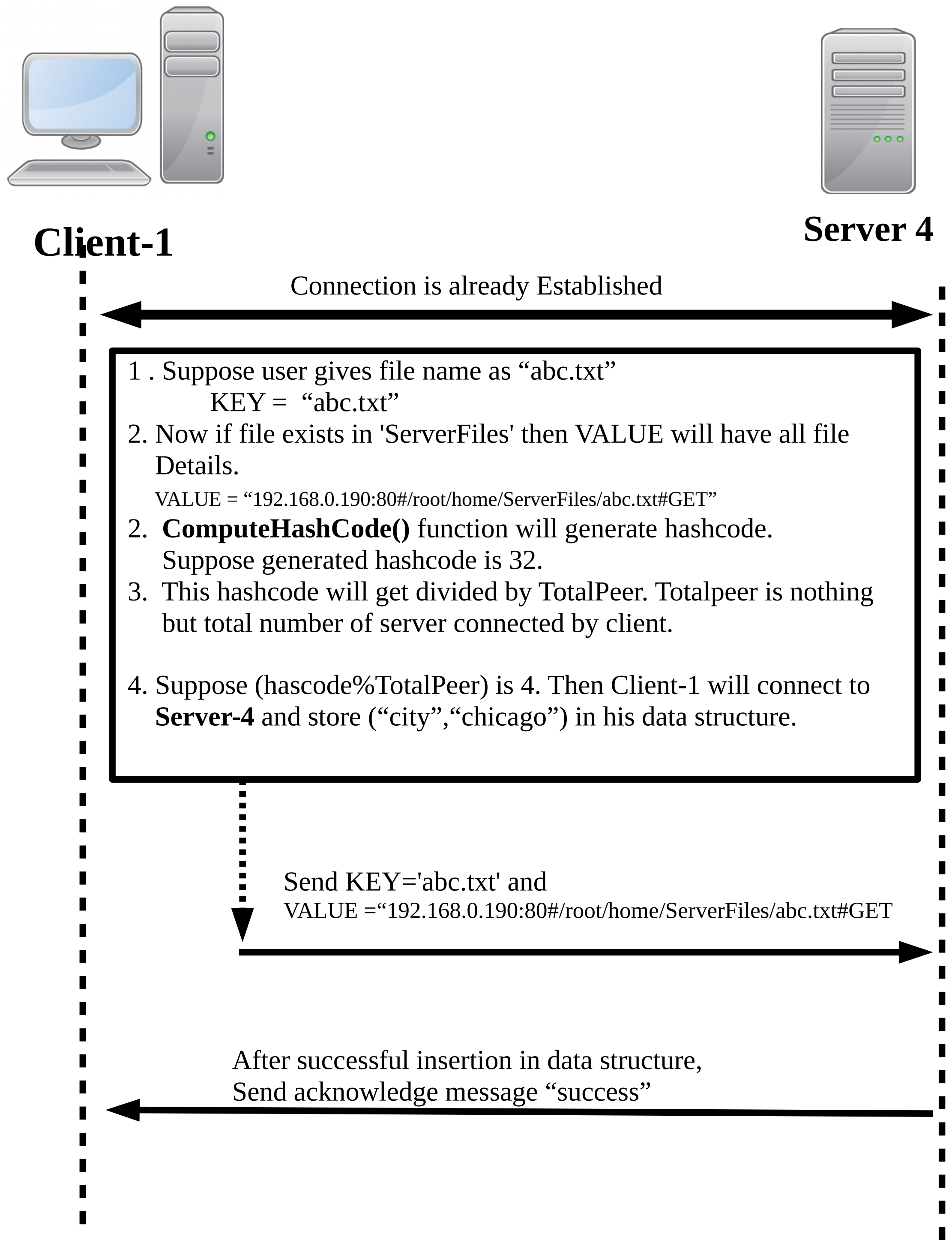
Step 1 : First Client-1 fetch information about all other server's IP address and PORT number from **'config.property'** file

Step 2 : According to design, Client will connect to all other servers in the sequential manner. In this step, Client-1 will connect to Server-1.

Step 3 : Client-1 will connect to Server-2.

Step 9 : Client-1 will connect to Server-8.

4. Register request by client :



1 . Suppose user gives file name as “abc.txt”
KEY = “abc.txt”

2. Now if file exists in 'ServerFiles' then VALUE will have all file Details.
VALUE = “192.168.0.190:80#/root/home/ServerFiles/abc.txt#GET”

2. **ComputeHashCode()** function will generate hashcode. Suppose generated hashcode is 32.

3. This hashcode will get divided by TotalPeer. Totalpeer is nothing but total number of server connected by client.

4. Suppose (hascode%TotalPeer) is 4. Then Client-1 will connect to **Server-4** and store (“city”,“chicago”) in his data structure.

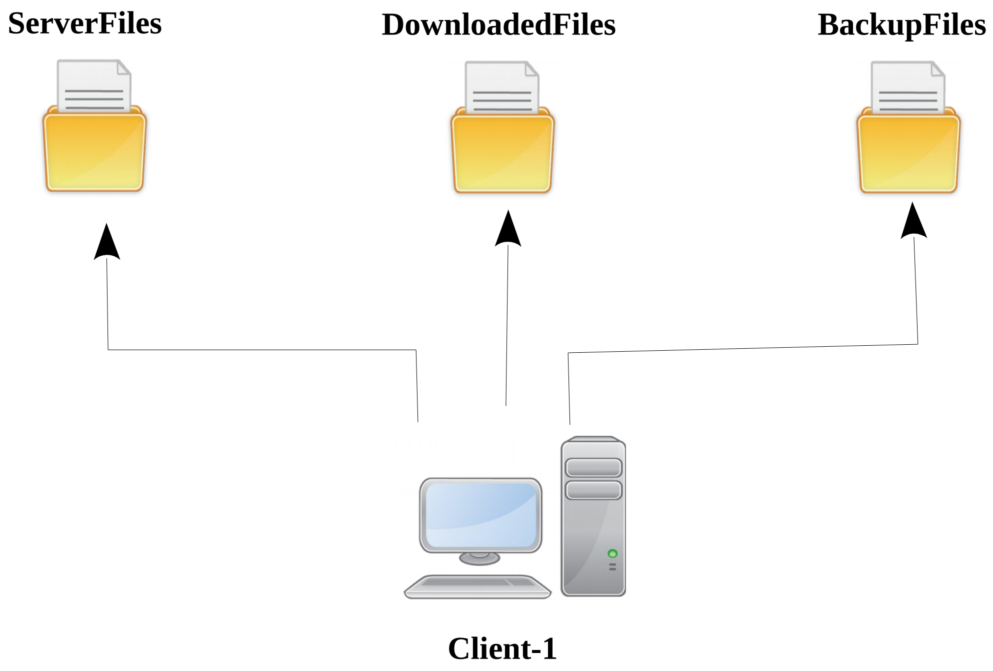
Send KEY='abc.txt' and
VALUE =“192.168.0.190:80#/root/home/ServerFiles/abc.txt#GET

After successful insertion in data structure,
Send acknowledge message “success”

5. Replica Scenario :

When User want to register a file, then replica of this file gets created at adjacent server.
Example:

1. If user want to register "ABC.txt" file.
2. Generated hashcode is 5.
3. Now file is going to register in SERVER_5.
4. Replica of file is going to save in SERVER_4.
5. All replica files is going to save in BackupFiles.
6. Actual server files will be at ServerFiles. So there will not be any conflict between actual files and server files.
7. Indexing server is going to handle replica request by "BACKUP_GET".
   (Code present in PeerServer_Instance.java)
8. Replica file gets downloaded in another server by making request to indexing server.
   IndexingServer will get replica download request as "DOWN".
   (Code present in PeerServer_Instance.java)



6. Condition where code will not work :

i. To increase performance, all messages are passing to-and from server/client in String format.

ii. Delimiter technique is used to distinguish GET/PUT/DELETE/DOWN/BACKUP_GET operation.

iii. If user give KEY = "ABC.txt" and VALUE = "chicago" for "PUT" operation,
then String that will pass to server is as follows :

"ABC.txt@@192.168.0.1:80/root/client1/src/ServerFiles/ABC.txt##BACKUP_GET"

iv. As shown above "@@" and "##", this two delimiter are used.

v. The code will not work, if user gives input which contains "@@" or "##" values.

7. Future Scope :

i. To overcome above delimiter limitation, String will be append with a header. This header will gives information about length of key and value. And last few bytes will be allocate for operation type.

ii. So above (KEY,VALUE) pair (“ABC.txt”,”ABC.txt@@192.168.0.1:80/root/client1/src/ServerFiles/ABC.txt##BACKUP_GET”) will be stored as

Length of KEY	Length of VALUE	Space allocated for KEY	Space allocated for VALUE	OPERATION CODE

OPERATION CODE :

000 = PUT
001 = GET
010 = DELETE
011 = BACKUP_GET
100 = DOWN
111 = INVALID