**PROGRAMMING ASSIGNMENT 3 – CS 519**

**DHT-BASED KEY-VALUE STORE**

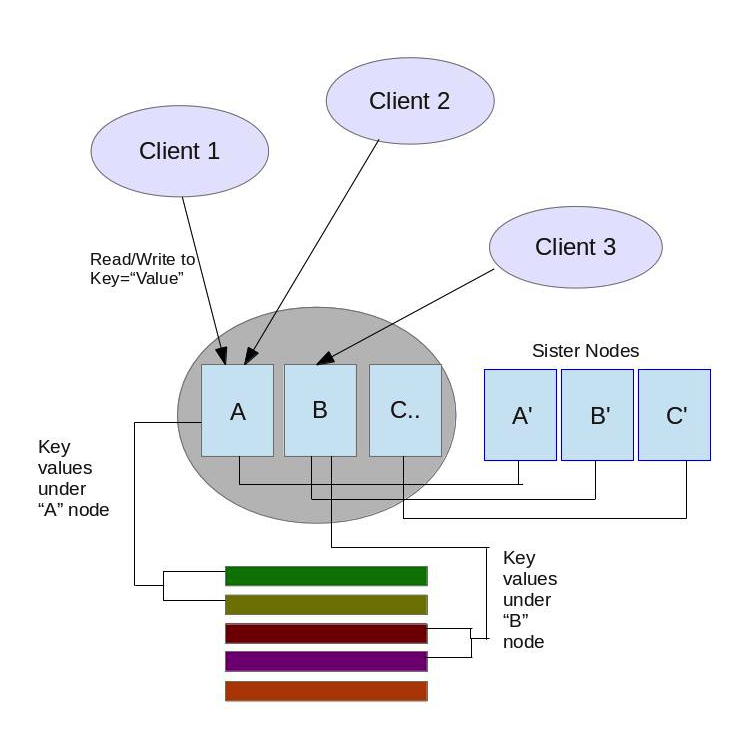
**Thekkada Mandanna, Karan Desai**

**INTRODUCTION**

The aim is to provide a look up service based on key=>value pairs that are stored in a Decentralized distributed environment. Any node can retrieve the value from the key. The DHT is maintained by primary nodes, that provide the main routing of request and executing them. These primary nodes have their own secondary/sister nodes which provide replication and load sharing.

**DESIGN**

This is an implementation of a DHT where there are a total of *n* server nodes. Half of them are primary nodes and the remaining half are the secondary nodes. The clients can send request to any of the primary nodes. Clients gives a Key and information regarding whether to write something to it or read from it. Based on the hashed value of these keys, they are distributed amongst the primary nodes. The client request is forwarded to the node that actually has the key, that processes it and returns back values or acknowledges. Apart from the primary nodes there are equal number of replicated secondary nodes. These secondary nodes are used for replication and for load sharing. The *write* requests are duplicated on every write request from the client. Note that the abstraction is always maintained and the client only gets back response from the node to which it had initially requested.



**IMPLEMENTATION**

For the programming of the system C++ and Unix Socket for Network Communication is used. The hashing algorithm is MD5. The following are the main procedures that are implemented in the process

**Connect**

One needs to specify the hostname to connect to that specific server. Once connected to the server the client can proceed to read/write “values” to the key.

For example – ***arkanoid.rutgers.edu or jfk.rutgers.edu***

**Read**

Read request from Client consists of the Key(file name). The full file is being read on this request

The read request looks like :

***0 [key]***

0 – Signifies Read

Key – the key (the file name) from where the value is to be read

**Write**

Write request consists of the key name and when prompted for the value, the client can enter what it wants to write to that key. Note that this will replace the old key values with the new ones.

The write request looks like

***1 [key] [values]***

1- signifies write operation

Key – the key/filename where the operation is to be performed

Values – Client can enter the values to be written the specific key

**Disconnect** simply disconnects the client from that host

**The key=>values**

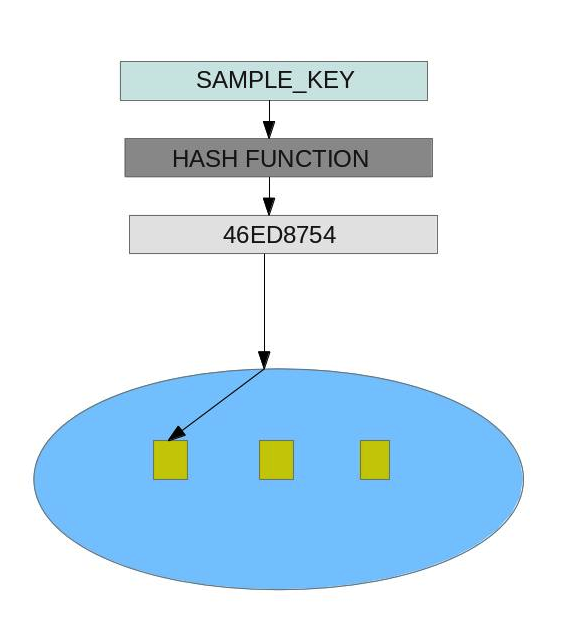
These key value pairs are maintained in terms of *files.*  When a user gives a “key” it is hashed to make a *filename,* which is stored with its corresponding primary node depending on the hash value range. The MD5 algorithm is being used for Hashing.

Example – If there are three primary nodes (A, B, C) then

A has the first one third [0- n/3]

B has the second [n/3 -2n/3]

C has the remaining one third [2n/3 – n]



**The Primary and Secondary Nodes**

The number of primary nodes can be changed by the integer value in the ***num\_of\_serv*** file.

Example: The contents of the file for primary nodes is just “3”

The current server number can be provided by the file ***serv\_id***

Example: “1”, “2” or “3”

The ***behind\_server*** and ***ahead\_server*** are used to link the servers. The hostnames of the server are provided in these files.

Example: *behind\_server* contents “jfk.rutgers.edu”

The secondary node for a specific primary node is indicated by the file ***secondary\_server***. The hostname of the secondary/sister node can be provided in this file

Example: *secondary\_server* contains cd.rutgers.edu

The primary node execution file is ***mult\_serv.cpp*** while the secondary node execution file is ***sis\_serv.cpp***

The clients can be executed by using the file ***client.cpp***

The **dht.h** file defined some common functions and the **md5.h** is for the MD5 hashing algorithm.

Every write request to primary node is also sent to its secondary node.

To maintain a load parameter for the primary node is maintained in the file ***load\_rate***. The primary node changes it depending on the number of clients that are connected. Another file ***max\_load\_rate***  maintains the load threshold for the primary node. If this limit is reached then the primary node will forward **only read requests** to the secondary nodes which performs the operation and sends back output to the primary nodes which redirect it to the client.

**STEPS INVOLVED IN PROCESS**

1. All servers - primary and secondary nodes to be setup first

(All files stated in the preceding section have to be setup for the nodes)

1. Client node initiated
2. Client node connects to Primary Node by Hostname
3. It gives the read/write command
4. Client waits for server response
5. Server side- Hash the key by using the MD5 algorithm
6. Find out the node (A, B, C…) to which the key belong to
7. Forward to request to that particular node
8. If read – fetch and return via the chain of nodes as request was received

10. Acknowledge the nodes about the write success/failure

11. Meanwhile on the sister nodes- If write is called on the primary node – perform the same operation on the secondary node.

12. Also if the request rate is high on the primary node forward the requests to the server

13. Client receives response.

14. End – Disconnect from Host

**INSTRUCTIONS TO RUN THE DHT:**

**On the primary server side**

1. Edit the **serv\_id** number to indicate the id of the current server where it is installed.

Eg:- If it is the first server to come up on the network , then value in the file should be 1.

The numbers should be unique and in ascending order.

2. Edit the **num\_of\_serv** file to indicate the total number of primary servers on the network.

3. Edit the **ahead\_server** file with the domain name of the next server to come up on the network:

Eg:- arkanoid.rutgers.edu

4. Edit the **behind\_server** file with the domain name of the previous server to come up on the network.

Eg:- oxr.rutgers.edu

5. Edit the **dht.h** file on line 18 to indicate required **keyspace**

6. Compile the server program as follows:

**g++ mult\_serv.cpp -o mult\_serv**

7. Run the server program with any port no from 20000 to 65535 but it should be identical on all

other servers and clients.

**./mult\_serv 50000**

**On the secondary server side**

1. The **serv\_id** file's contents must be the same as it's primary server.

2. Compile the **sis\_serv.cpp** program:

**g++ sis\_serv.cpp -o sis\_serv**

3. Run the sister node:

**./sis\_serv 50000**

**On the client side**

1. Compile the client side program

**g++ client.cpp -o client**

2. Run the client program

**Eg:- ./client 50000**

Check for different scenarios.

**CONCLUSION**

The requirements of the implemented system as stated on the problem statement are achieved. The same has been tested for multiple scenarios and the results we obtained as per requirement.

References:

<http://en.wikipedia.org/wiki/Distributed_hash_table>

<http://en.wikipedia.org/wiki/Consistent_hashing>

<http://www.zedwood.com/article/121/cpp-md5-function>

<http://bobobobo.wordpress.com/2010/10/17/md5-c-implementation/>

[http://en.wikipedia.org/wiki/Chord\_%28peer-to-peer%29](http://en.wikipedia.org/wiki/Chord_(peer-to-peer))

<http://www.linuxhowtos.org/C_C++/socket.htm>

**Note:-** Screenshots have been included to in the .tar file to show to how to run the programs.