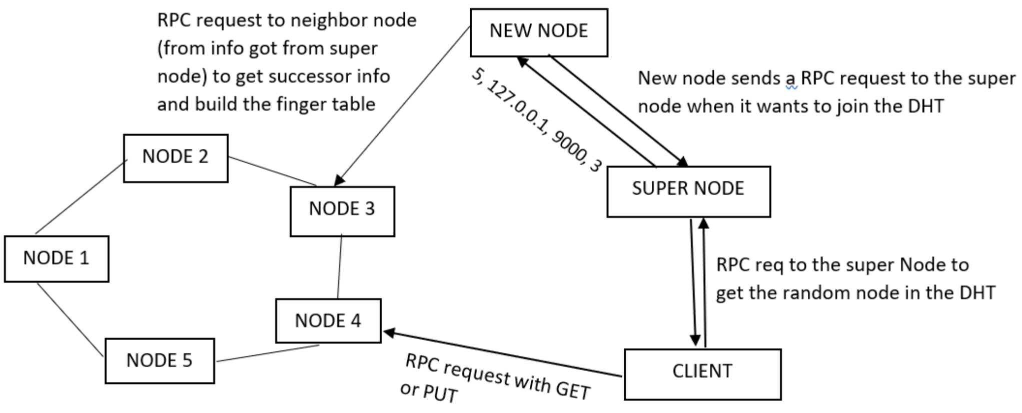
INTRODUCTION TO DISTRIBUTED SYSTEMS (CSCI 5105)
PROGRAMMING ASSIGNMENT-2

DESIGN DOCUMENT

GROUP MEMBERS:
Sai Pratyusha Attanti *(attan005@umn.edu, Student ID: 5656785)*
Navya Ganta *(ganta016@umn.edu, Student ID: 5673223)*

SYSTEM DESIGN:


RPC request to neighbor node (from info got from super node) to get successor info and build the finger table
NEW NODE
5, 127.0.0.1, 9000, 3
New node sends a RPC request to the super node when it wants to join the DHT
NODE 2
NODE 3
NODE 1
SUPER NODE
RPC req to the super Node to get the random node in the DHT
NODE 4
NODE 5
RPC request with GET or PUT
CLIENT

Here we will be implementing a dictionary using a DHT system. It is a peer-to-peer system where all the nodes have the same power. We have a SuperNode that keeps track of the nodes joining the DHT and acts as the first node client contacts in the system to access the dictionary.
Initially, a DHT network will be built with the nodes who want to join the network one by one. Once the network is built, the client can use the DHT network to put or set the word into the dictionary. All the connections are synchronous in the system.

SuperNode.py:
This file contains the implementation of the Super Node. It is the first node to run in the system and regulates the nodes joining the DHT, and also the first node the client contacts in the system.

If a node wants to join the network it makes the RPC to the supernode with the information about its IP address and the port. If the Super Node is already busy with the other node, the client has to wait till the supernode becomes free. If not, it accepts the request from the new node and creates a unique identifier(node ID) for that node(using hashing), and gives it the information about one of the nodes in the DHT to which it can contact to join the network in this format '127.0.0.1, 8000, 12, 3' (where 127.0.0.1, 8000, 12 is the IP address, port and the ID of the random node respectively and 3 is the ID generated for the new node).

Similarly when a client wants to access the dictionary, it makes the RPC to the supernode and gets the information about one of the nodes in DHT to which it can contact. It returns the information to the client in this format 127.0.0.1, 8000, 12 (where 127.0.0.1, 8000, 12 is the IP address, port, and the ID of the random node respectively)

SuperNode doesn't keep any information about the system except the list of nodes in the network.

Node.py:
This file contains the implementation of our Nodes in the DHT. So when a node is ready to join the network, it sends the RPC call to the supernode and gets the information about the random node in the DHT. So the new node contacts the node it received information about and with the help of this node it joins the network. When a New node joins the network, It keeps the information about its successor, predecessor, creates a new finger table for itself, and updates all the other nodes' finger tables. And it consoles its node information after the node has joined the network or its finger table is updated.
Every node maintains a dictionary data structure to store the meanings of the word. Which node will store the data will depend on the hash value generated for that word. If a node gets a request to set or get a word, first it checks if the current node is responsible for storing that word. If so, it stores the word in the dictionary in case of set and fetches the word meaning in case of get. If the node is the node responsible for that word, it forwards the request to the other based on its finger table data. The request is forwarded until it reaches the node responsible for the word.

Client.py:
This file contains the implementation of our client. Once the DHT is ready, the client can contact the supernode and get the information about one of the nodes in the DHT to which it can contact. That node is responsible for creating a hash for the word(making sure the words and the nodes fall in the same keyspace). We used SHA1 hashing to hash the word.
So when a client contacts this node, it takes control from there, it hashes the word and using the finger table it finds the node which contains the meaning or node where it can put the new word(in case of a put).
List of operations the client can perform: 1. set, 2. get 3. getNodes 4. exit
Based on the command line input, it can perform the respective operations.

1. set lets the user set the meaning of the word into the dictionary. The word and its meaning for the parameters for this operation.
2. get allows the user to get the meaning of the word from the dictionary and the parameter for this operation is the word to which we want to fetch the meaning.
3. getNodes fetches the information about all the nodes in the DHT and the words, meanings they are storing
4. exit operation closes the client.

The client is also allowed to set the meanings of the multiple words by providing the file name as the input while running the client.
Also, the Client consoles the path followed to reach the node storing the word.

CACHE IMPLEMENTATION:
To reduce the lookup time we are using caching in our system. The main idea here is to maintain multiple copies of word meaning pairs in different nodes to reduce the lookup time. For this, we are maintaining a dictionary at each node and multiple copies of the pair are inserted into all the nodes which were accessed while executing the set operation.
Here we are assuming that the word meaning will not be updated once set into the DHT. If the client tries to set the word which is already present in the dictionary, set operations with an error message.

CONFIGURATION:
This config.cfg file stores the necessary information required to configure the system dynamically.

- Addresses and port numbers of the client, supernode, and the nodes in the DHT.
- The number of nodes in the DHT. (default number of nodes: 5)
- The number of bits to determine the keyspace (default bits: 5)

DATA:
The sample dictionary file (dictionary_sample.txt) is provided in the project folder which can be used to set the meanings of multiple words at one time. (If you want to use a different file make sure the data format matches with the dictionary_sample.txt)

INSTRUCTIONS:
To run this project we will need Python 3 and Thrift.

- Make sure to modify the env.txt file according to the location of libraries in your system and make sure to add the project directory to the path.
- Check the config.cfg file to make sure all the addresses and port numbers of the nodes in the systems are as desired.
- Also, check the config.cfg if other information is as desired.
- Instructions to run:
    1. Run '**python3 SuperNode.py**' to run the Super Node.
    2. To run the node we need to provide a number as a command-line argument, which will be used to fetch the address and port from the config.cfg file on which the node should run.
       The number starts from 0 and should be within the DHT size.
       Run '**python3 Node.py i**' to run the node.
       Example: 'python3 Node.py 0' to run the first node(so based on the index provided here(i.e 0) address and port will be fetched.)
    3. Run '**python3 Client.py**' to run the Client.
       If you want to set the multiple words at one time:
       Run `**python3 Client.py dictionary_sample.txt**`
       (If you want to use a different file make sure the data format matches with the dictionary_sample.txt)

TEST CASES:

1. If the client requests the meaning of the word that is present in the DHT dictionary, the word meaning is fetched and is displayed in the client console along with the path it followed to reach the node storing the word.
2. If the client sets the meaning to a word in the dictionary, the word is hashed and is sent to the respective node to store the word. Once the word is set successfully, a success message is displayed in the client console along with the path it followed to reach the node.

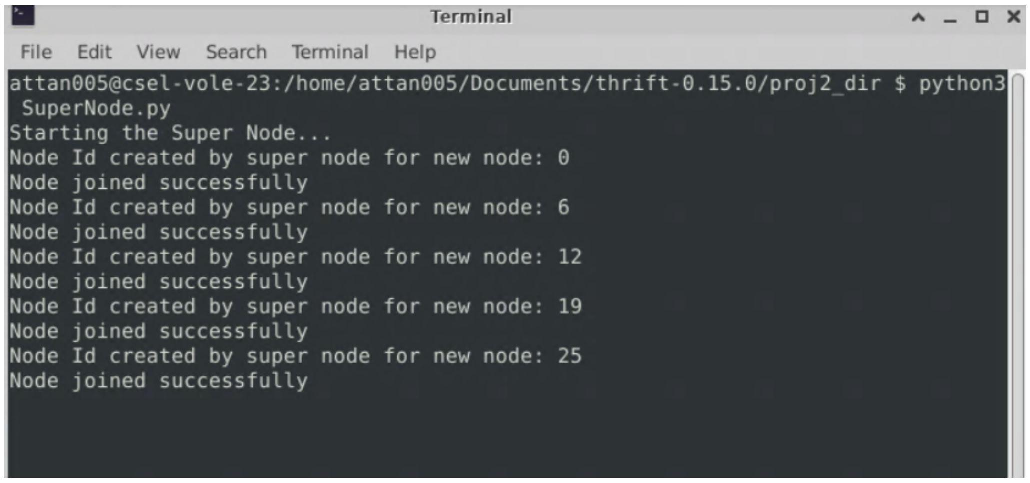
Negative Test cases considered:

1. If the Super Node is busy in the join process of a node, it rejects all the other node requests by sending the "NACK" to the new node and prints the error message "Rejected join request"

2. Since the minimum number of nodes in the DHT should be 5. If provided DHT size in the config file is less than 5, supernode exits with an error message " DHTSize should be greater than 4, please update the config file"
3. If the Client contacts the supernode before the DHT is built, the client's request to the DHT is rejected and the supernode sends a negative acknowledgment "NACk" with the error message "DHT join still in progress"
4. If the client requests a word that is not present in the dictionary, the error message is displayed in the client console "Entered word meaning not found"
5. If the client tries to set the meaning of the word already present in the DHT dictionary, the set operation fails with an error message "The word is already present in the dictionary. Please try another word."

SAMPLE RESULTS:

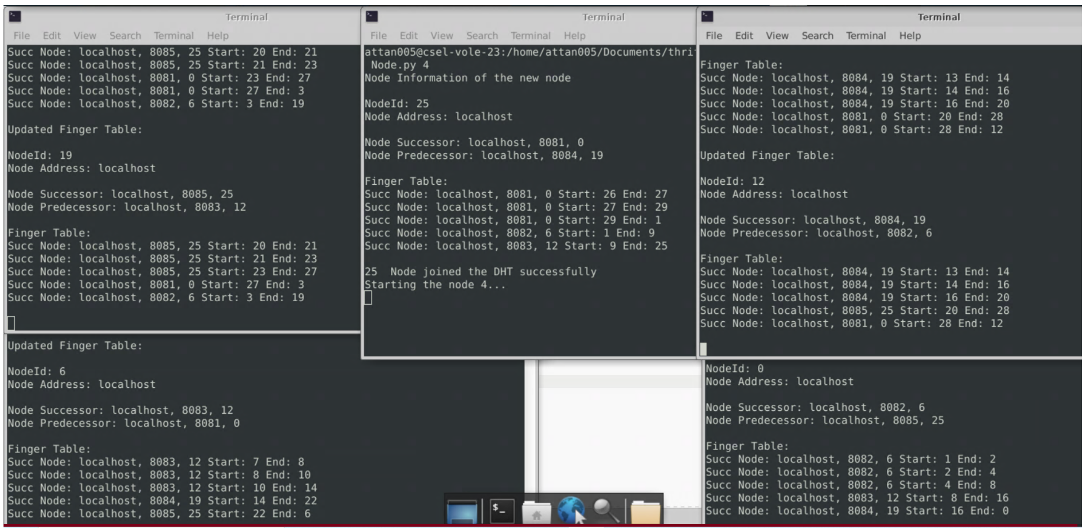
Screenshot of working Super Node:



```
attan005@csel-vole-23:/home/attan005/Documents/thrift-0.15.0/proj2_dir $ python3
 SuperNode.py
Starting the Super Node...
Node Id created by super node for new node: 0
Node joined successfully
Node Id created by super node for new node: 6
Node joined successfully
Node Id created by super node for new node: 12
Node joined successfully
Node Id created by super node for new node: 19
Node joined successfully
Node Id created by super node for new node: 25
Node joined successfully
```

As the communication is synchronous, the supernode waits till the node joins the network.

Screenshot of Nodes in the DHT:



```
Succ Node: localhost, 8085, 25 Start: 20 End: 21
Succ Node: localhost, 8085, 25 Start: 21 End: 23
Succ Node: localhost, 8081, 0 Start: 23 End: 27
Succ Node: localhost, 8081, 0 Start: 27 End: 3
Succ Node: localhost, 8082, 6 Start: 3 End: 19

Updated Finger Table:

NodeId: 19
Node Address: localhost

Node Successor: localhost, 8085, 25
Node Predecessor: localhost, 8083, 12

Finger Table:
Succ Node: localhost, 8085, 25 Start: 20 End: 21
Succ Node: localhost, 8085, 25 Start: 21 End: 23
Succ Node: localhost, 8085, 25 Start: 23 End: 27
Succ Node: localhost, 8081, 0 Start: 27 End: 3
Succ Node: localhost, 8082, 6 Start: 3 End: 19
```

```
attan005@csel-vole-23:/home/attan005/Documents/thri
 Node.py 4
Node Information of the new node

NodeId: 25
Node Address: localhost

Node Successor: localhost, 8081, 0
Node Predecessor: localhost, 8084, 19

Finger Table:
Succ Node: localhost, 8081, 0 Start: 26 End: 27
Succ Node: localhost, 8081, 0 Start: 27 End: 29
Succ Node: localhost, 8081, 0 Start: 29 End: 1
Succ Node: localhost, 8082, 6 Start: 1 End: 9
Succ Node: localhost, 8083, 12 Start: 9 End: 25

25  Node joined the DHT successfully
Starting the node 4...
```

```
Finger Table:
Succ Node: localhost, 8084, 19 Start: 13 End: 14
Succ Node: localhost, 8084, 19 Start: 14 End: 16
Succ Node: localhost, 8084, 19 Start: 16 End: 20
Succ Node: localhost, 8081, 0 Start: 20 End: 28
Succ Node: localhost, 8081, 0 Start: 28 End: 12

Updated Finger Table:

NodeId: 12
Node Address: localhost

Node Successor: localhost, 8084, 19
Node Predecessor: localhost, 8082, 6

Finger Table:
Succ Node: localhost, 8084, 19 Start: 13 End: 14
Succ Node: localhost, 8084, 19 Start: 14 End: 16
Succ Node: localhost, 8084, 19 Start: 16 End: 20
Succ Node: localhost, 8085, 25 Start: 20 End: 28
Succ Node: localhost, 8081, 0 Start: 28 End: 12
```

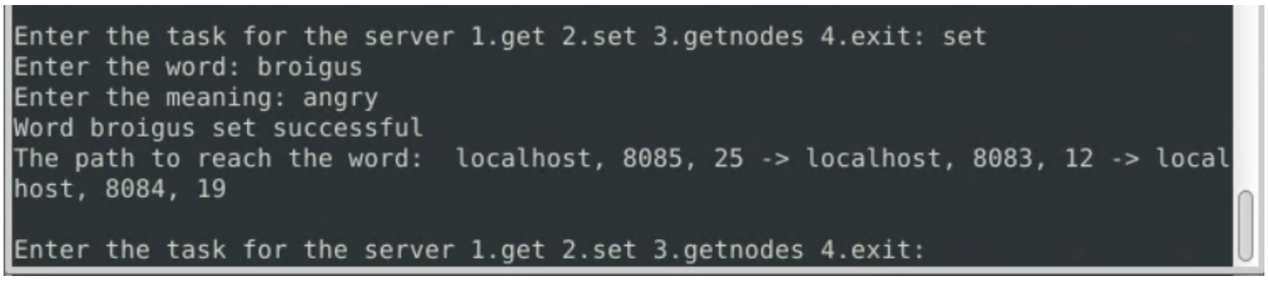
```
Updated Finger Table:

NodeId: 6
Node Address: localhost

Node Successor: localhost, 8083, 12
Node Predecessor: localhost, 8081, 0

Finger Table:
Succ Node: localhost, 8083, 12 Start: 7 End: 8
Succ Node: localhost, 8083, 12 Start: 8 End: 10
Succ Node: localhost, 8083, 12 Start: 10 End: 14
Succ Node: localhost, 8084, 19 Start: 14 End: 22
Succ Node: localhost, 8085, 25 Start: 22 End: 6
```

```
NodeId: 0
Node Address: localhost

Node Successor: localhost, 8082, 6
Node Predecessor: localhost, 8085, 25

Finger Table:
Succ Node: localhost, 8082, 6 Start: 1 End: 2
Succ Node: localhost, 8082, 6 Start: 2 End: 4
Succ Node: localhost, 8082, 6 Start: 4 End: 8
Succ Node: localhost, 8083, 12 Start: 8 End: 16
Succ Node: localhost, 8084, 19 Start: 16 End: 0
```

Each node consoles its node information after the node has joined the network or its finger table is updated.

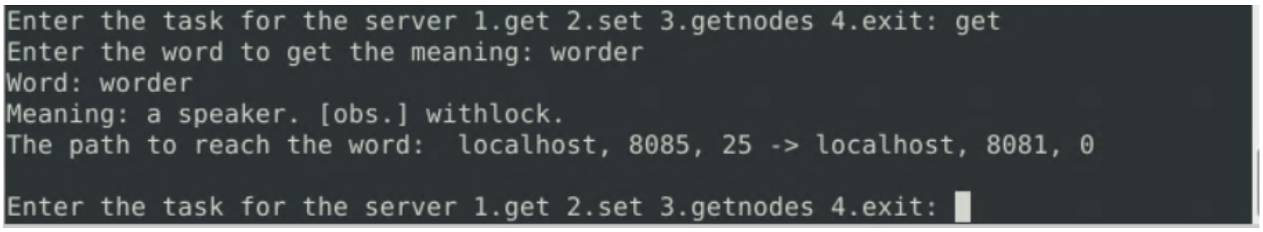
Screenshot of the client:

set word: (sets the word and the meaning into the dictionary)



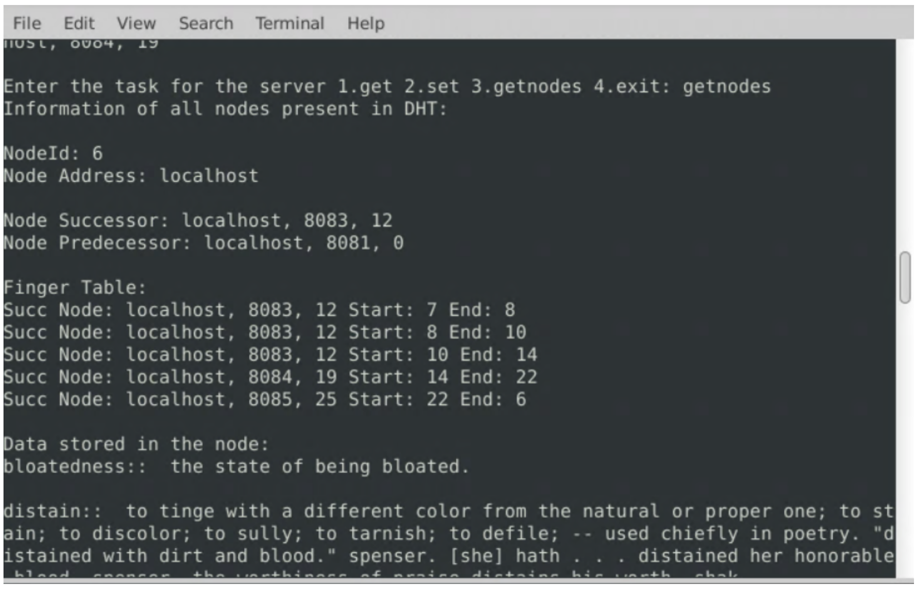
```
Enter the task for the server 1.get 2.set 3.getnodes 4.exit: set
Enter the word: broigus
Enter the meaning: angry
Word broigus set successful
The path to reach the word:  localhost, 8085, 25 -> localhost, 8083, 12 -> local
host, 8084, 19

Enter the task for the server 1.get 2.set 3.getnodes 4.exit:
```

get word: (gets the requested word meaning)



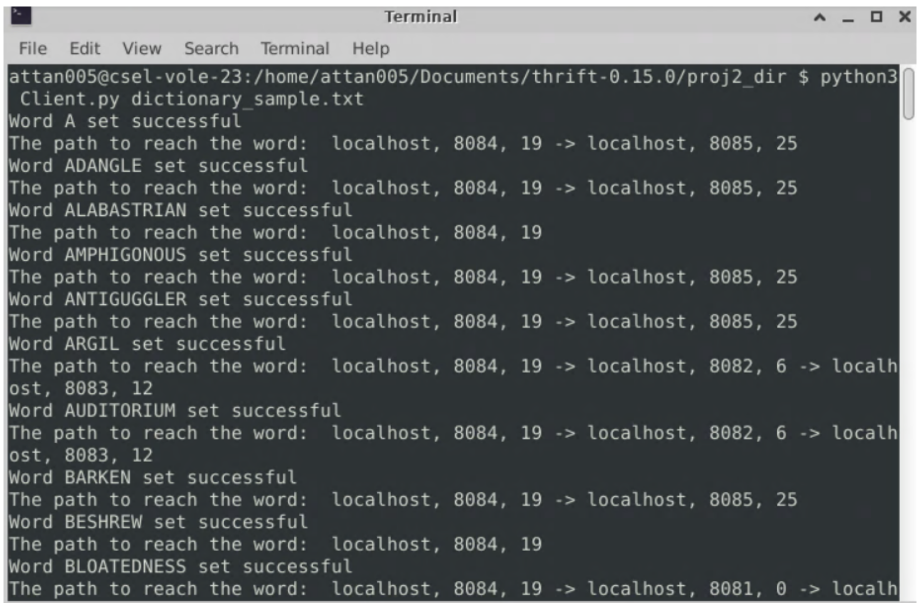
```
Enter the task for the server 1.get 2.set 3.getnodes 4.exit: get
Enter the word to get the meaning: worder
Word: worder
Meaning: a speaker. [obs.] withlock.
The path to reach the word:  localhost, 8085, 25 -> localhost, 8081, 0

Enter the task for the server 1.get 2.set 3.getnodes 4.exit: 
```

getNodes information: (gets the information about the nodes in the network along with the data stored in the nodes)



```
host, 8084, 19

Enter the task for the server 1.get 2.set 3.getnodes 4.exit: getnodes
Information of all nodes present in DHT:

NodeId: 6
Node Address: localhost

Node Successor: localhost, 8083, 12
Node Predecessor: localhost, 8081, 0

Finger Table:
Succ Node: localhost, 8083, 12 Start: 7 End: 8
Succ Node: localhost, 8083, 12 Start: 8 End: 10
Succ Node: localhost, 8083, 12 Start: 10 End: 14
Succ Node: localhost, 8084, 19 Start: 14 End: 22
Succ Node: localhost, 8085, 25 Start: 22 End: 6

Data stored in the node:
bloatedness::  the state of being bloated.

distain::  to tinge with a different color from the natural or proper one; to st
ain; to discolor; to sully; to tarnish; to defile; -- used chiefly in poetry. "d
istained with dirt and blood." spenser. [she] hath . . . distained her honorable
 blood. spenser. the worthiness of praise distains his worth. shak
```

Setting multiple words at a time into the dictionary:



```
attan005@csel-vole-23:/home/attan005/Documents/thrift-0.15.0/proj2_dir $ python3
 Client.py dictionary_sample.txt
Word A set successful
The path to reach the word:  localhost, 8084, 19 -> localhost, 8085, 25
Word ADANGLE set successful
The path to reach the word:  localhost, 8084, 19 -> localhost, 8085, 25
Word ALABASTRIAN set successful
The path to reach the word:  localhost, 8084, 19
Word AMPHIGONOUS set successful
The path to reach the word:  localhost, 8084, 19 -> localhost, 8085, 25
Word ANTIGUGGLER set successful
The path to reach the word:  localhost, 8084, 19 -> localhost, 8085, 25
Word ARGIL set successful
The path to reach the word:  localhost, 8084, 19 -> localhost, 8082, 6 -> localh
ost, 8083, 12
Word AUDITORIUM set successful
The path to reach the word:  localhost, 8084, 19 -> localhost, 8082, 6 -> localh
ost, 8083, 12
Word BARKEN set successful
The path to reach the word:  localhost, 8084, 19 -> localhost, 8085, 25
Word BESHREW set successful
The path to reach the word:  localhost, 8084, 19
Word BLOATEDNESS set successful
The path to reach the word:  localhost, 8084, 19 -> localhost, 8081, 0 -> localh
```