

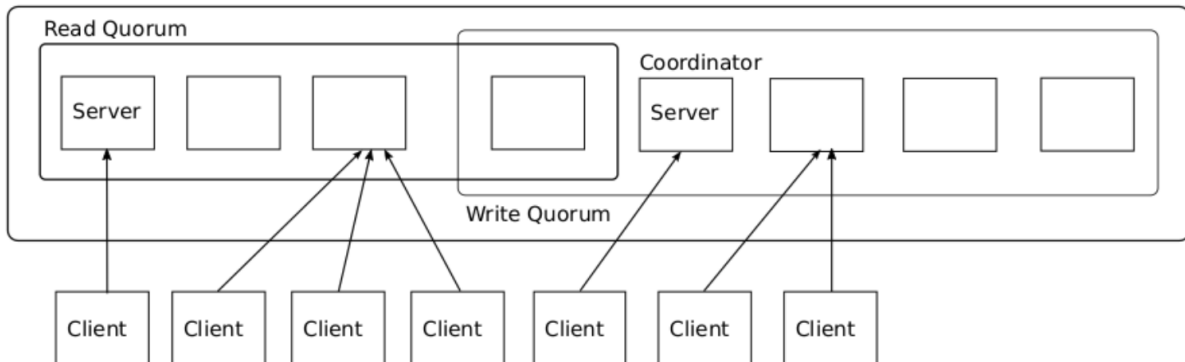
INTRODUCTION TO DISTRIBUTED SYSTEMS (CSCI 5105)
PROGRAMMING ASSIGNMENT-3
DESIGN DOCUMENT

GROUP MEMBERS:

Sai Pratyusha Attanti (attan005@umn.edu, Student ID: 5656785)

Navya Ganta (ganta016@umn.edu, Student ID: 5673223)

SYSTEM DESIGN:



Here we implemented a simple distributed file system in which multiple clients can share files using the quorum-based protocol. It is a peer-to-peer system where all the nodes have the same power. One of the file servers will act as the coordinator for your quorum, so when a write/read request is sent to a node it is forwarded to the coordinator to carry out that operation.

All the files are replicated across all the nodes. So when a request is forwarded to the coordinator it chooses random nodes as quorum nodes based on the sizes specified for read quorum (N_R) and write quorum (N_W) in config.cfg file. Before serving any read/write request we make sure quorum constraints are satisfied.

1. $N_W + N_R > N$
2. $N_W > N/2$

Now the read/write operations will be performed on the respective quorum nodes.

Node.py:

This file contains the implementation of the file servers. One among these nodes acts as the coordinator for the system to carry out the operations.

Read Operation:

When the client sends a request to read a file to one of the randomly chosen node in the system, the request is forwarded to the coordinator node which randomly chooses the read quorum nodes of size N_R to return the contents from node having the latest version of the file, i.e It contacts the read quorum nodes, and get the node with the latest version and fetches the most up-to-date information from that node and returns it to the client.

If the file is not present in the servers, it returns an error message to the client.

Write Operation:

When the client sends a write request for a file to one of the randomly chosen node in the system, the request is forwarded to the coordinator node which randomly chooses the write quorum nodes of size N_w and gets the latest version of the file by contacting these write quorum nodes. And now the file is updated with the new data in all the write quorum nodes and the version of this file is now updated to latest_version+1 in all these nodes. Once the write is complete it is notified to the client. If the file is not present in the server, we create a new file and write the content to this file.

Since the files can be accessed concurrently by the multiple client, during the write operation the file to which the write is performed is locked to ensure that the consistency is maintained and for sequentially ordering concurrent writes to the same file. So when a file receives a concurrent write request from a different client, it has to wait until the previous lock on this file is released.

List Files:

When the client request to list all the files with their latest version to one of the randomly chosen node in the system, the request is forwarded to the coordinator node which randomly chooses the read quorum nodes of size N_r and contacts them to the latest version of each file and return them to the client.

Client.py :

This file contains the implementation of the client. If a client wants to read/write a file, it randomly chooses one file server and sends an RPC request to that node, and waits for the result.

List of operations client can perform are 1. read 2. write 3. list files 4. writeall 5. readall

Based on the command line input, it can perform the respective operations.

1. read: Read operation reads the contents of the file. The parameters for this operation is the file name from which we want to read the content. And the return value for this operation is the contents of the file.
2. write: Write operation updates/writes the contents of the file. The parameters for this operation is the file name, and the contents we want to replace the file with. The return value for this operation is a boolean value indicating whether the write is successful or not.
3. list files: This operation is mainly for testing purposes. It fetches all the files and their latest versions by contacting read quorum nodes. The output of this operation is list of the files and their latest version numbers.

For load testing our system with different combinations of N_r and N_w , we have added an additional feature to inject heavy read and heavy write by simultaneously sending read and write requests to all the files. This can achieved by running 'writeall' for firing write requests simultaneously and 'readall' for firing read requests simultaneously in the client.

CONFIGURATION :

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

- Addresses and the port numbers of all the nodes
- Coordinator Node information
- Number of servers in the system-node_count (default node_count : 7)
- Quorum size for reads and write. (default N_r : 6, N_w : 4)
- Path to the files (files/) and the replica files(/tmp/files).

DATA:

The sample data for this project are present in the files/ directory.

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.
- Make sure to add the project directory to the path.
- Check the config.cfg file to make sure all the ports and other options are as desired.
- Also, check the config.cfg if other information is as desired.
- Instructions to run:

1. 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 node_count(i.e number of file servers in the system).

Run '**python3 Node.py i**' to run the node.

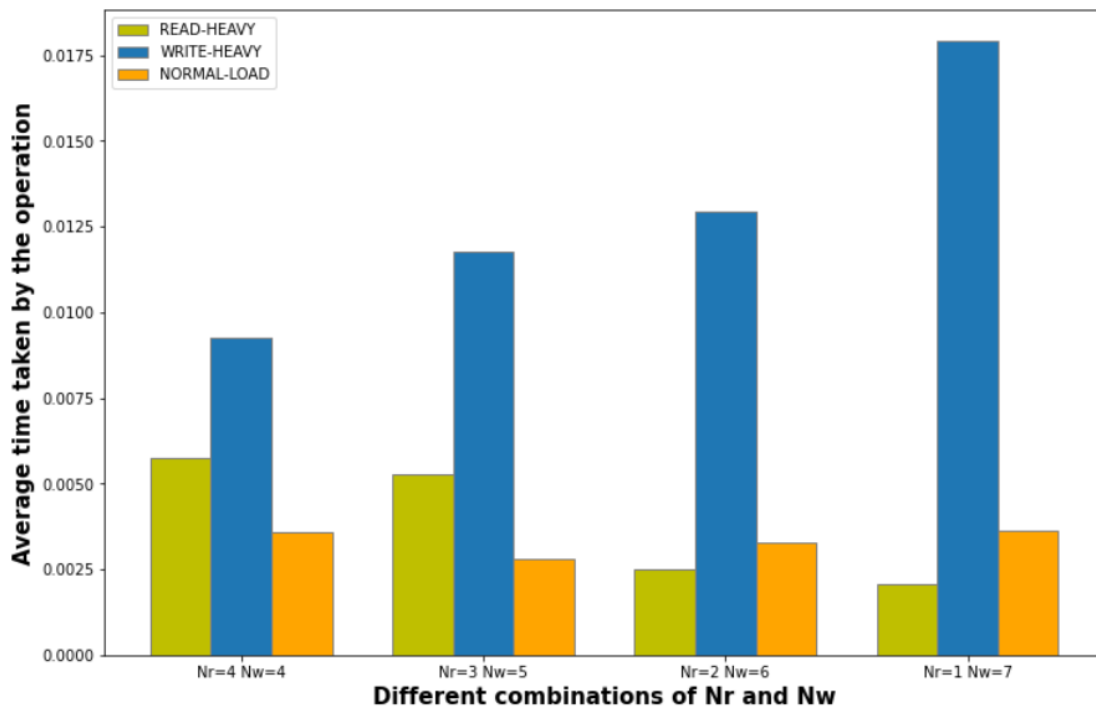
Example: '**python3 Node.py 0**' to run the first node(so based on the index provided here(i.e 0) address and port will be fetched.)

2. Run '**python3 Client.py**' to run the Client

PERFORMANCE ANALYSIS:

We have tested the performance of our system with combinations of the N_R and N_W under different workloads. For the analysis we made sure to consider the edge-cases like read-heavy system, right-heavy system, and if the requested file not present in the server.

Below is the graph plotted between the combination of different N_R and N_W values and the time taken to complete the operation under different workloads in the system:



Insights from the above graphs:

- From the above graph we see that in the read-heavy workload case, as the value of N_R decreases, the average time taken to complete the task also decreases gradually because since the number of nodes in the read quorum reduces, the number of nodes contacted to get the latest content from the file also increases which leads to low job completion time. So in a read-heavy system it is advantageous to have less value for N_R .
- Similarly, In the write-heavy case as the value of N_W increases, the the average time taken to complete the task also increases gradually because since the number of nodes in the write quorum increases, the number of nodes contacted to update the file in replicas also increases which leads to high job completion time.

Negative test cases considered for performance analysis:

1. For Read operation, if the file is not present in any of the servers, we return an error message “File Not Present” to the client.
2. For write operation, if the file is not present in the server, we create a file and write the content to that file, and update its version to the latest version of the file.
3. For the given values of N_R and N_W if the quorum constraints are not satisfied, it exits with an error message.

SAMPLE RESULTS:

node_count = 7

$N_R = 4$ and $N_W = 4$

For Write operation:

Screenshot of working system:

```
attan005@cse1-vole-45:/home/attan005/Documents/thrift-0.15.0/proj3_dir $ python3 Node.py 0
Starting the node 0...
Write request to the file : file2.txt received from client
Coordinator received the write request to the file: file2.txt
Selected nodes for writing to the file file2.txt are:
['localhost, 8086', 'localhost, 8082', 'localhost, 8084', 'localhost, 8081']
Writing to the file: file2.txt

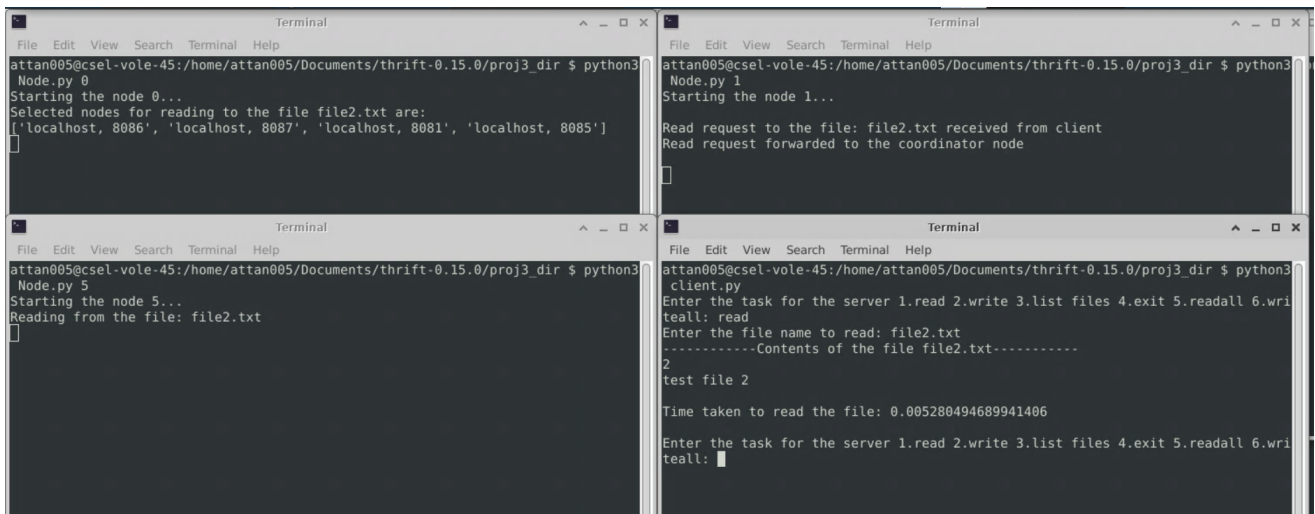
attan005@cse1-vole-45:/home/attan005/Documents/thrift-0.15.0/proj3_dir $ python3 Node.py 1
Starting the node 1...
Writing to the file: file2.txt

attan005@cse1-vole-45:/home/attan005/Documents/thrift-0.15.0/proj3_dir $ python3 Node.py 3
Starting the node 3...
Writing to the file: file2.txt

attan005@cse1-vole-45:/home/attan005/Documents/thrift-0.15.0/proj3_dir $ python3 client.py
Enter the task for the server 1.read 2.write 3.list files 4.exit 5.readall 6.writeall: write
Enter the file name to write: file2.txt
Enter the content of the file: testing write operation
write operation completed successfully
Time taken to write the file: 0.005515098571777344
Enter the task for the server 1.read 2.write 3.list files 4.exit 5.readall 6.writeall: 
```

For Read operation:

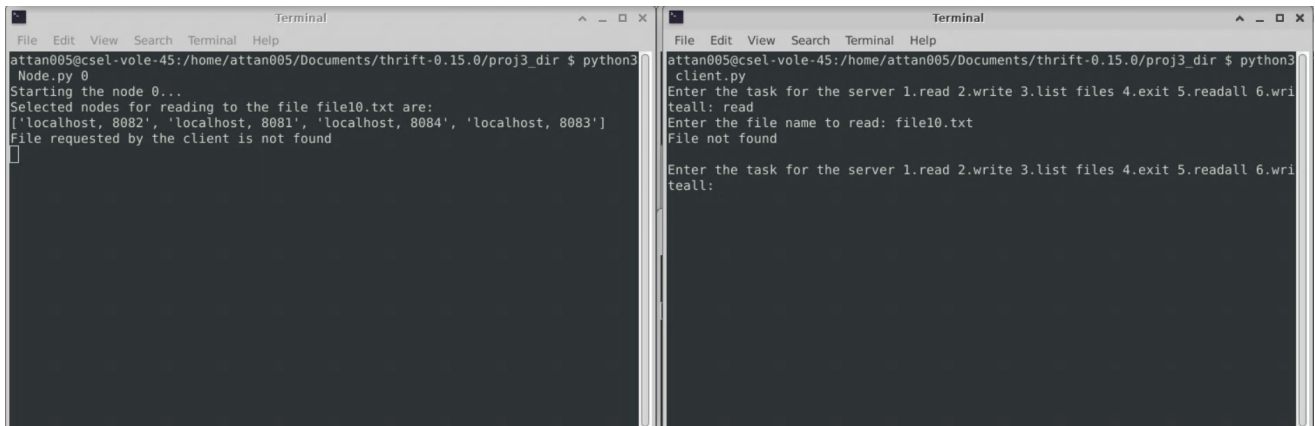
Screenshot of working system:



The screenshot displays four terminal windows illustrating the read operation workflow:

- Terminal 1 (Node 0):** Shows the command `python3 Node.py 0` and the output: "Starting the node 0...", "Selected nodes for reading to the file file2.txt are: ['localhost, 8086', 'localhost, 8087', 'localhost, 8081', 'localhost, 8085']".
- Terminal 2 (Node 1):** Shows the command `python3 Node.py 1` and the output: "Starting the node 1...", "Read request to the file: file2.txt received from client", "Read request forwarded to the coordinator node".
- Terminal 3 (Node 5):** Shows the command `python3 Node.py 5` and the output: "Starting the node 5...", "Reading from the file: file2.txt".
- Terminal 4 (Client):** Shows the command `python3 client.py` and the output: "Enter the task for the server 1.read 2.write 3.list files 4.exit 5.readall 6.wri", "teall: read", "Enter the file name to read: file2.txt", "-----Contents of the file file2.txt-----", "2", "test file 2", "Time taken to read the file: 0.005280494689941406", "Enter the task for the server 1.read 2.write 3.list files 4.exit 5.readall 6.wri", "teall: ".

If the file is not present in the system:

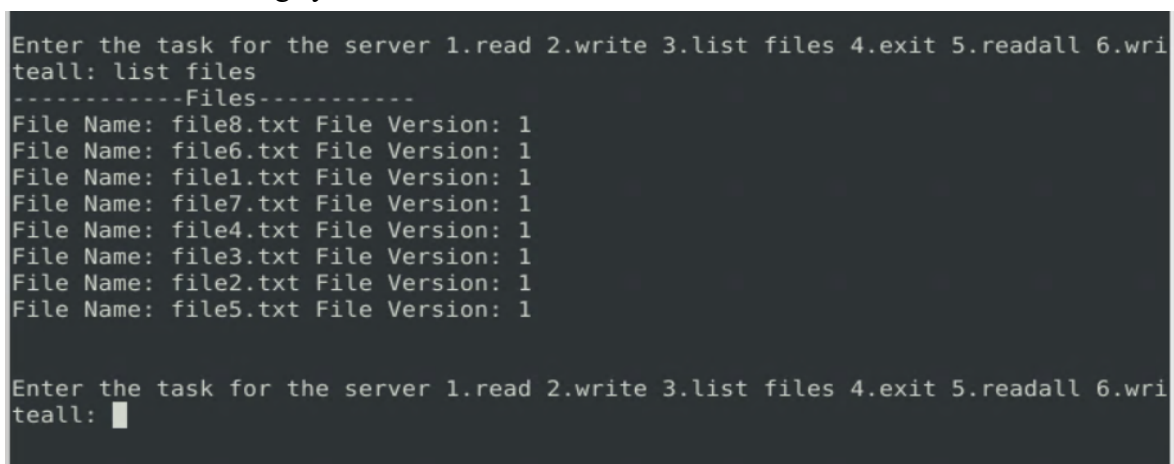


The screenshot displays two terminal windows illustrating the error handling for a missing file:

- Terminal 1 (Node 0):** Shows the command `python3 Node.py 0` and the output: "Starting the node 0...", "Selected nodes for reading to the file file10.txt are: ['localhost, 8082', 'localhost, 8081', 'localhost, 8084', 'localhost, 8083']", "File requested by the client is not found".
- Terminal 2 (Client):** Shows the command `python3 client.py` and the output: "Enter the task for the server 1.read 2.write 3.list files 4.exit 5.readall 6.wri", "teall: read", "Enter the file name to read: file10.txt", "File not found", "Enter the task for the server 1.read 2.write 3.list files 4.exit 5.readall 6.wri", "teall: ".

For List All files:

Screenshot of working system:



The screenshot displays a terminal window showing the list all files operation:

```
Enter the task for the server 1.read 2.write 3.list files 4.exit 5.readall 6.wri
teall: list files
-----Files-----
File Name: file8.txt File Version: 1
File Name: file6.txt File Version: 1
File Name: file1.txt File Version: 1
File Name: file7.txt File Version: 1
File Name: file4.txt File Version: 1
File Name: file3.txt File Version: 1
File Name: file2.txt File Version: 1
File Name: file5.txt File Version: 1

Enter the task for the server 1.read 2.write 3.list files 4.exit 5.readall 6.wri
teall: 
```