# Programming Assignment 2: Simple Distributed File System

## Execution Steps

**Java Version -** 1.7.0_76

**Thrift Version -** 0.9.3

**Compilation Steps:**

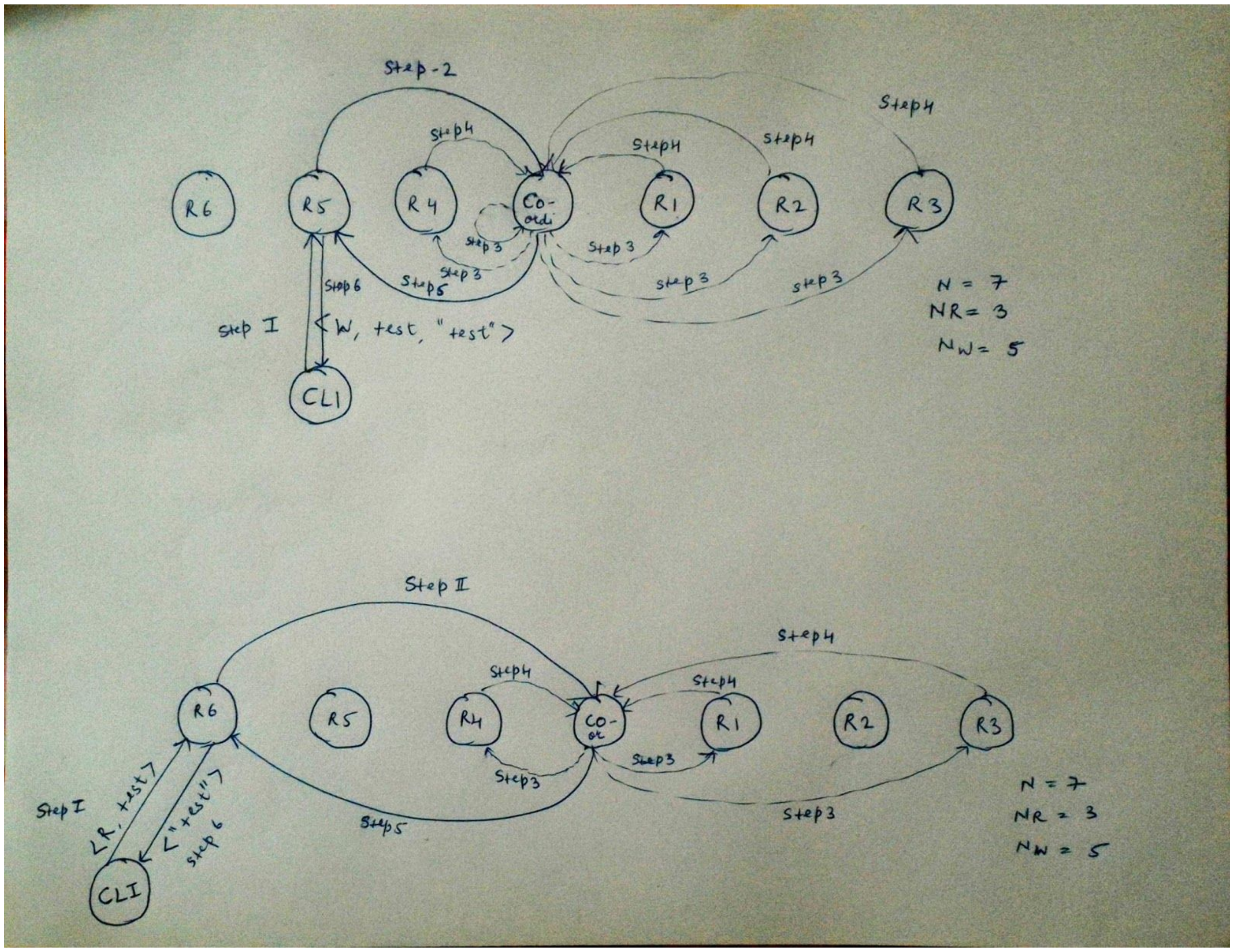
1) *javac -cp .:<thrift_jars_directory>/* FileServer.java*
2) *javac -cp .:<thrift_jars_directory>/* Client.java*

**Steps to make Distributed System up and running:**

1) *java -cp .:<thrift_jars_directory>/* FileServer <coordinator_ip> <coordinator_port_no> <server_port_no> <NR> <NW>*
2) *repeat Step 1 for k-1 number of times, where k = number of servers you want to add in DFS.*
3) *java -cp .:<thrift_jars_directory>/* Client <server_ip> <server_port_no> <operation> <file_name> <contents>*
   *operation can be W (write), R (read), I (get DFS Information), S (synch)*
   *All the parameters are mandatory for Write.*
   *<contents> is not required for Read.*
   *<file_name>, <contents> are not required for DFS Information and Synch*

## Design Architecture

Our System Design has the following main modules/components :



**Read/Write Quorum Based Algorithm :**

In above Figure, we have taken total replicas in system ( including coordinator) as 7. Read Quorum, NR is 3 and Write Quorum, NW as 5. “Co-” represents coordinator, “CLI” represents client and R[1-6] represent replicas other than coordinator.

Roles:

a) Coordinator - It is one of the replicas, which is selected/elected to play role of coordinator. In our system implementation we are selecting first replica to work as a coordinator as well.Some of the functions of Coordinator are :
   i) It runs synch process in background after every n second.
   ii) It is responsible for collecting appropriate quorum from other replicas.
   iii) All read/write request from other replicas will be directed to coordinator who will gather Quorums based on the corresponding operations and responds accordingly to these replicas.
   iv) It can also accept read/write request from clients similar to other replicas.

v) It maintains values of NR and NW and also when system setup is complete, it is responsible for checking whether current system is in consistent state or not based on these conditions :
    1) 0 < NR =< N
    2) 0 < NW =< N
    3) NR+NW > N
    4) NW > N/2

b) Client - Client makes read and write request from any of the available replicas. With the help of client, we can also trigger synch operation. We can also check the information about all files currently residing in system along with their current version number at different replicas.
c) Replica - Replica is used to increase availability, fault tolerance and to reduce latency. Some of the functionality of Replica Servers are :
    i) It receives request from Client and connect to coordinator to perform Read/Write request
    ii) It maintains copies of files that it has along with version number into its memory and provides this information to coordinator whenever requested.

Operations:

a) Read Operation : Client will issue read command from any given replicas. Request will go from the replica to Coordinator, if the replica is not coordinator, otherwise it will skip this step. Coordinator will gather NR Quorums from replicas, and based on highest version number, it will return file content to requested replica, which further replies to client. If file is not there, it will return an error.
b) Write Operation : Client will issue write command from any given replicas. Request will go from the replica to Coordinator, if the replica is not coordinator, otherwise it will skip this step. Coordinator will first gather NW Quorums from replicas, and based on highest version number, it will update its content and increment its version number. If file is not there, it will create a new file with version 1. After this step, coordinator will update this file with content of latest version gathered in previous step for NW Quorums.
c) Synch Operation : After every 30 seconds, Coordinator will run synch operation in background, whose main objective is to bring every replica into consistent state eventually. In this operation, we are first assembling a read quorum, and getting the details (name, content and version) of all the files present on these quorum. Using these details, we are generating the most updated file version for all the files. These updated versions are then pushed to all the replicas.

**Thrift File**

Thrift file contains the service - FileServerService, and it exposes the following methods :

- synch
- write
- read
- getSystemInfo
- updateCoordinator
- writeFromCoord
- readFromCoord
- getFileMapFromCoord

**Assumptions**

We have assumed that system will run in stages. In first stage, we will make our coordinator up. In next stage, we will make all our replicas up. And finally, in last stage, clients will issue read and write operations from different replicas. We are building one service in thrift namely FileServerService and we are treating both replicas and coordinator as same. Read Quorum, NR and Write Quorum, NW are passed to coordinator before starting any individual experiment. These values cannot be changed when a particular experiment is running but it can be changed across different experiments. Also, once a replica is there in system, it will be there forever and there is zero probability of its failure. We have not stored file on disk and it is always there in memory in form of map. Map is simple with string key and string value. Key will be the name of the file, and value will contain contents and version separated by a predefined separator. All read and write will go through coordinator. All replicas in system knows IP address of the coordinator. If either of NR or NW is greater than total number of replicas, N, we have treated it as a case of an error condition. While we were taking readings on machines of cse labs, server was getting overloaded. This may be due to fact that other groups were also trying to do same. As a result we feel some of our readings might not be consistent. One more assumption is regarding Version number of file. It always starts with Version 1 and increments by 1 on every subsequent write operation. We have also assumed that all read operations can run concurrently, when there is no write in system. However, writes are serialized. If there is a write operation triggered by client, it will wait for all currently running reads to complete. Moreover, it will also block all read and write operation until it gets complete. One important property to note here is that we have used fairness property of lock, which means that during waiting, whatever request ( Read/Write) comes first, lock will be granted to it. So, as explained in class forum, if request appears like this:

Read(1) Read(2) Read(3) Write(1) Read(4) Read(5) Write(2) Read(6)

First, Read(1), Read(2), Read(3) gets executed concurrently, after which Write(1) and after write completes, Read(4) and Read(5) will execute concurrently, after which Write(2) and on completion of this write, finally Read(6) will get executed.

**Test Cases**

NR: Read Quorum

NW: Write Quorum

N: Total Number of replicas in system

a) NR and NW are same and equal to number of replicas, N
b) NR is 1 and NW is equal to number of replicas
c) NR is half number of replicas and NW is equal to half of number of replicas
d) NR is equal to number of replicas and NW is equal to N/2+1.
e) All Replicas are on same machine and client on remote location
f) All Replicas are on same machine and client on same location
g) All Replicas and client are on different machines
h) Read/Write operations coming concurrently ( To ensure this, we used sleep in Coordinator for testing)
i) Read/Write coming Sequentially
j) Doing all reads/write on same file
k) Doing read/write on mixed files

**Mixed Jobs - Here we have equal number of read/write jobs**

l) Number of read operations and write operations are same
m) Number of read operations and write operation are in proportion of 3:4
n) Number of read operations and write operation are in proportion of 4:3

**Read Intensive Requests - Here we kept read jobs high and write jobs low**

o) Number of read operations are high and there is only one write operation
p) Number of read operations and write operation are in proportion of 1:4

**Write Intensive Requests - Here we kept write jobs high and read jobs low**

q) Number of write operations are high and there is only one read operation
r) Number of write operations and read operation are in proportion of 1:4

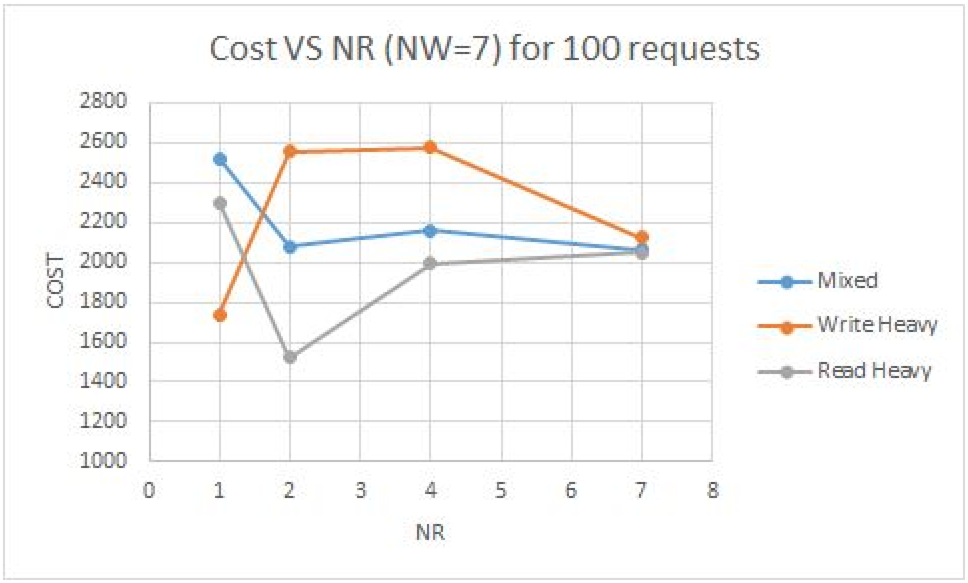
**Negative Test Cases:**

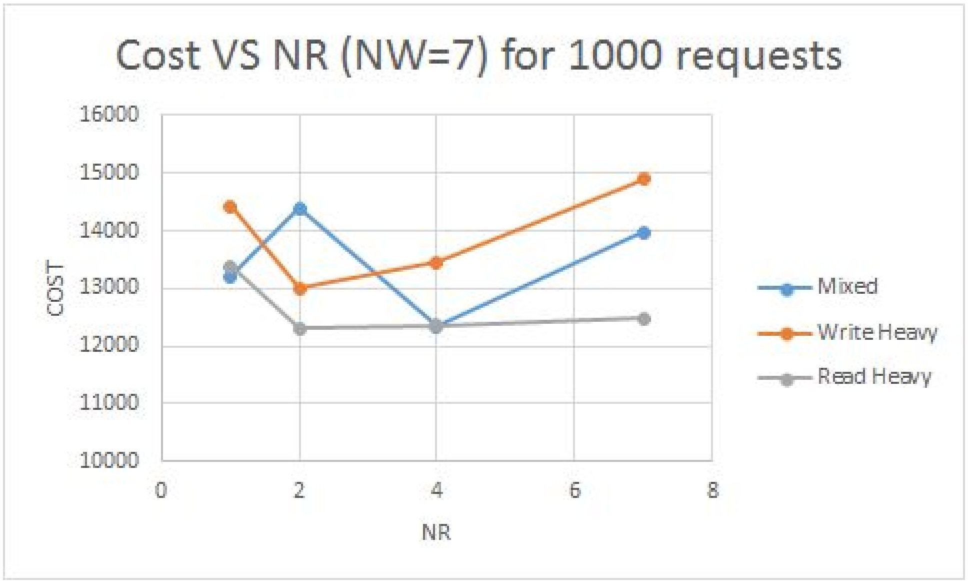
a) When Firewall is enabled which is blocking port number on client or replicas.
b) When we are using special port numbers such as 8080
c) When we are using port number which are already in use
d) When passing wrong IP address
e) When passing wrong port number
f) If coordinator is down and we are trying to add replica in system
g) If coordinator is down and client is trying read/write operations
h) If there is no replica from which client is trying to read/write
i) Client is trying to read file which does not exist

**Result:**
We successfully implemented Quorum based Distributed file system using thrift. We tried different Read and write Quorums and different job mix.

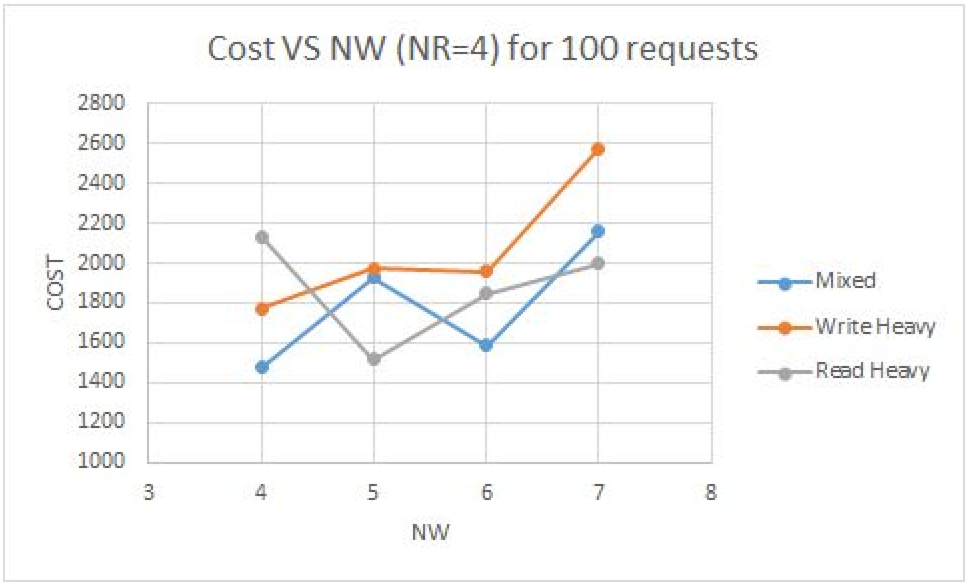
We executed the following test runs -

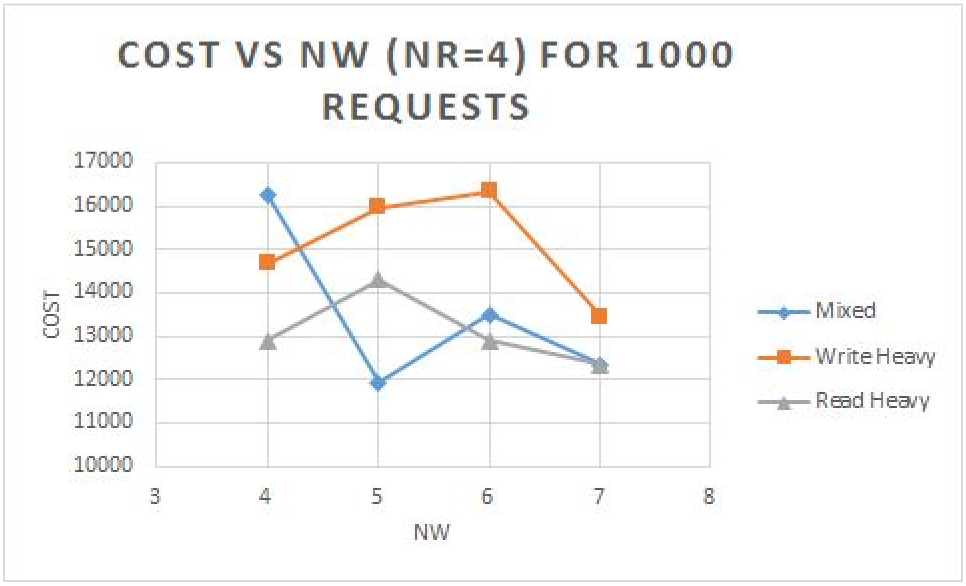
1) Keep NW constant, and vary NR. This setup was run for different job mixes - read heavy load, write heavy load, and mixed load. Please find below plots for the data collected for these runs for 100 and 1000 requests.





2) Keep NR constant, and varied NW. This setup was run for different job mixes - read heavy load, write heavy load, and mixed load. Please find below plots for the data collected for these runs for 100 and 1000 requests.





Expected Results -

1) Cost should vary directly as NW increases, keeping other parameters constant
2) Cost should vary directly as NR increases, keeping other parameters constant
3) Cost should vary directly as number of requests increases, keeping other parameters constant
4) Cost for different job mixes should be in the following order - read-heavy, mixed, write-heavy, keeping other parameters constant

We observed that in general ( from graph and from table in appendix), as the number of NR and NW increases, cost increases because the system has to sent more network requests to gather more number of Quorums before read/write can be completed. Also, read heavy requests are less costly for lower values of NR. Reason being, since majority of jobs are read only, and so if read Quorum is lower, majority of requests need to gather only small amount of Quorum, which in turn results in lower cost. Similarly, write-heavy requests are less costly for lower values of NW. And as expected, the cost increases with the number of requests. The plots do not completely comply with the expected results due to the following reasons -

1) Varying network traffic
2) During some runs, synch process may also have been executing in the background, and thus, increasing the cost a bit

**REFERENCES :**

1) Thrift RPC tutorial by Kwangsung Oh
2) Distributed Systems: Principles and Paradigms (2nd Edition)", by Andrew S. Tanenbaum and Maarten van Steen
3) Thrift Apache Documentation ( https://thrift.apache.org)
4) Stack OverFlow
5) ReentrantReadWriteLock - (https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantReadWriteLock.html)

## Appendix

For Request : 100

| NR | Mixed | Write Heavy | Read Heavy |
|---|---|---|---|
| 1 | 2520 | 1740 | 2299 |
| 2 | 2080 | 2554 | 1520 |
| 4 | 2160 | 2575 | 1995 |
| 7 | 2065 | 2123 | 2049 |

| NW | Mixed | Write Heavy | Read Heavy |
|---|---|---|---|
| 4 | 1474 | 1770 | 2134 |
| 5 | 1925 | 1973 | 1516 |
| 6 | 1587 | 1956 | 1847 |

| | | | |
|---|---|---|---|
| 7 | 2160 | 2575 | 1995 |

For 1000 requests

| NR | Mixed | Write Heavy | Read Heavy |
|---|---|---|---|
| 1 | 13219 | 14425 | 13387 |
| 2 | 14397 | 13012 | 12309 |
| 4 | 12346 | 13462 | 12356 |
| 7 | 13984 | 14890 | 12477 |

| NW | Mixed | Write Heavy | Read Heavy |
|---|---|---|---|
| 4 | 16275 | 14676 | 12907 |
| 5 | 11938 | 15976 | 14314 |
| 6 | 13499 | 16335 | 12897 |
| 7 | 12346 | 13462 | 12356 |