Simple Distributed File System

Shrivardhan S. Bharadwaj(5478571)

Himanshu Kharkwal(5488345)

University of Minnesota

INDEX

1. Purpose

In this programming assignment, we are implementing a simple distributed file system in which multiple clients can share files together. In this file system, the files are replicated to several servers for increased performance and availability.

2. Method

2.1 System Design

We implemented three components:

1. Client : Several clients can send read/write requests at the same time to the file server.
2. File Server: The file server receives multiple jobs from multiple Clients as well as coordinator.
    A. File Server receives a job from Client to read/write a file into the system. The file servers maintains the replicas of the files.
    B. File Server forwards the requests to the coordinator.
    C. The coordinator contacts the other randomly chosen servers needed for the quorum to complete the operation.
3. Coordinator : The Coordinator execute tasks such as:
    A. Coordinator receives requests from multiple File Servers.
    B. The coordinator contacts the other randomly chosen servers needed for the quorum to complete the operation. The coordinator is already known.

2.2 List of Files

1. config : User should provide the readQuorum, writeQuorum, CoordinatorPort, CoordinatorIp, and servers.
2. Client.java :
    I. Asks the user to input file for a job.
    II. Generates different test cases for the user to test the system.
    III. Requests File Server to perform the job.

3. FileServer.java : It can run different file servers on a single machine on different ports.
4. CoordinatorServer.java : It validates the constraints on write and read file servers quorum size.
5. FileServiceHandler.java : receive jobs from a client, and sends them to Coordinator. Receive jobs from a coordinator.
6. QuorumServiceHandler.java : It receives sync, read and write operations. Contacts quorum and gets the latest versions of the file and returns max version to the server. It does not file store versions locally. It ensures that read operations on a file are parallel but write and sync operations are sequential.
7. VersionServiceHandler.java : Clients contacts it for read and write operations. It internally contacts the coordinator for latest version and returns it to the client. Periodically every 15 secs a sync operation is run by contacting the coordinator in background.
8. Make : Compiles all the java files
9. Run : Starts the framework
10. FileService.thrift
11. QuorumService.thrift
12. VersionService.thrift
13. Dataset folder : contains 9 text files.
14. TestCases folder : contains various test cases to evaluate the functioning of the system.

2.3 Logs generated

1. We are generating logs of sync operation on file servers. Each file server prints the latest version of all the replicas it contains.

   Format: After syncing

   ```
   {file1=version, file2=version,....,filen=version}
   ```
2. Prints out the latest versions of files for read and write operations.

Write operation logs format:

```
Version = updated version
Time taken: number of ms
```

Read operation logs format:

```
File Contents = file data
Version = updated version
Time taken: number of ms
```

3. How to run

1. SSH to client and file server machines:

   ```
   $ ssh <client>
   $ ssh <file server>
   $ ssh <coordinator>
   ```

2. Download and extract the tar to some location in client

   ```
   $ tar -xzf <tarfile>
   ```

3. Edit config file and add readQuorum, writeQuorum, CoordinatorPort, CoordinatorIp, and servers.

   ```
   $ nano config
   > readQuorum=Nr
   > writeQuorum=Nw
   > CoordinatorPort=Port Number
   > CoordinatorIp=IP_address
   > servers=IP_1:port_1,IP_2:port_2,....,IP_n:port_n
   ```

4. Execute Make file at client for compiling all thrift files and Java Code

   ```
   $ sh Make
   ```

5. Execute Run file at each machine to start the framework. Note: Start Coordinator and File Servers before starting the Client.

   In Coordinator machine:

   ```
   $ sh Run CoordinatorServer
   ```

In File Servers machines:

```
$ sh Run FileServer
```

In client machine:

```
$ sh Run Client
```

This will open a menu and prompt the user for the following inputs:

```
> Menu:
1: Read a file
2: Write a file
3: Execute Test Cases
4: Quit
```

If you click execute test cases it will open up another menu to choose test cases.

```
> Menu:
1: Write one file
2: Read one file
3: Write multiple files
4: Read multiple files
5: Read/Write multiple files
6: Write heavy one file
7: Read heavy one file
8: Write heavy multiple files
9: Read heavy multiple files
10: Multiple Read and Write one file
11: Multiple Read and Write multiple files
```

4. Test cases

Negative cases:

1. Case : When we try to read a file which has not been written yet.

Output : Prints “version=0” which signifies that no file server has the replica of that file.

Positive cases:

We have created a Test cases folder which includes all 11 positive test cases in 11 files.

Format of a case:

filename:operation<w/r>,number of times the operation being run on that file

1. Case : Write one file.

   Output : returns the updated version of the file.
2. Case : Read one file.

   Output : returns the most recent version of the file.
3. Case : Write multiple files. Different multiple files are being written.

   Output : writes multiple files and returns the updated version of each file.
4. Case : Read multiple files. Different multiple files are being read.

   Output : returns the most recent version of the multiple files.
5. Case : Read/Write multiple files. Different multiple files are being read/written.

   Output : returns the most recent/updated version of the multiple file.
6. Cases : Write heavy one file. One file is being written heavily(100 times) by many clients.

   Output : returns the updated version of the file to each client call.
7. Cases : Read heavy one file. One file is being read heavily(100 times) by many clients.

   Outputs : returns the most recent version of the file to each client call.
8. Cases : Write heavy multiple files. Different multiple files are being written heavily by many clients.

   Outputs : returns the updated version of the multiple files to each client call.
9. Cases : Read heavy multiple files. Different multiple files are being read heavily by many clients.

   Outputs : returns the most recent version of the multiple files to each client call.

10. Cases : Multiple Read and Write one file. One file is being written or read by many clients.
    Outputs : returns the most recent/updated versions of the multiple files.
11. Cases : Multiple Read and Write multiple files. Multiple file is being written or read by many clients.
    Outputs : returns the most recent/updated versions of the multiple files.

5. Performance Metrics

For the performance evaluation we have taken the following metrics:

Heavy read and write have 100 connections in parallel.

N = 7

1. Nr = 7, Nw = 4
   Heavy read time taken = 320ms
   Heavy write time taken = 1379ms
2. Nr = 6, Nw = 4
   Heavy read time taken = 316ms
   Heavy write time taken = 1336ms
3. Nr = 5, Nw = 4
   Heavy read time taken = 308ms
   Heavy write time taken = 1357ms
4. Nr = 4, Nw = 4
   Heavy read time taken = 296ms
   Heavy write time taken = 1305ms
5. Nr = 4, Nw = 5
   Heavy read time taken = 290ms
   Heavy write time taken = 1396ms
6. Nr = 4, Nw = 6
   Heavy read time taken = 297ms
   Heavy write time taken = 1397ms

7. Nr = 4, Nw = 7

   Heavy read time taken = 303ms

   Heavy write time taken = 1359ms

8. Nr = 3, Nw = 5

   Heavy read time taken = 286ms

   Heavy write time taken = 1324ms

9. Nr = 3, Nw = 6

   Heavy read time taken = 289ms

   Heavy write time taken = 1432ms

10. Nr = 3, Nw = 7

    Heavy read time taken = 285ms

    Heavy write time taken = 1358ms

11. Nr = 2, Nw = 6

    Heavy read time taken = 277ms

    Heavy write time taken = 1401ms

12. Nr = 2, Nw = 7

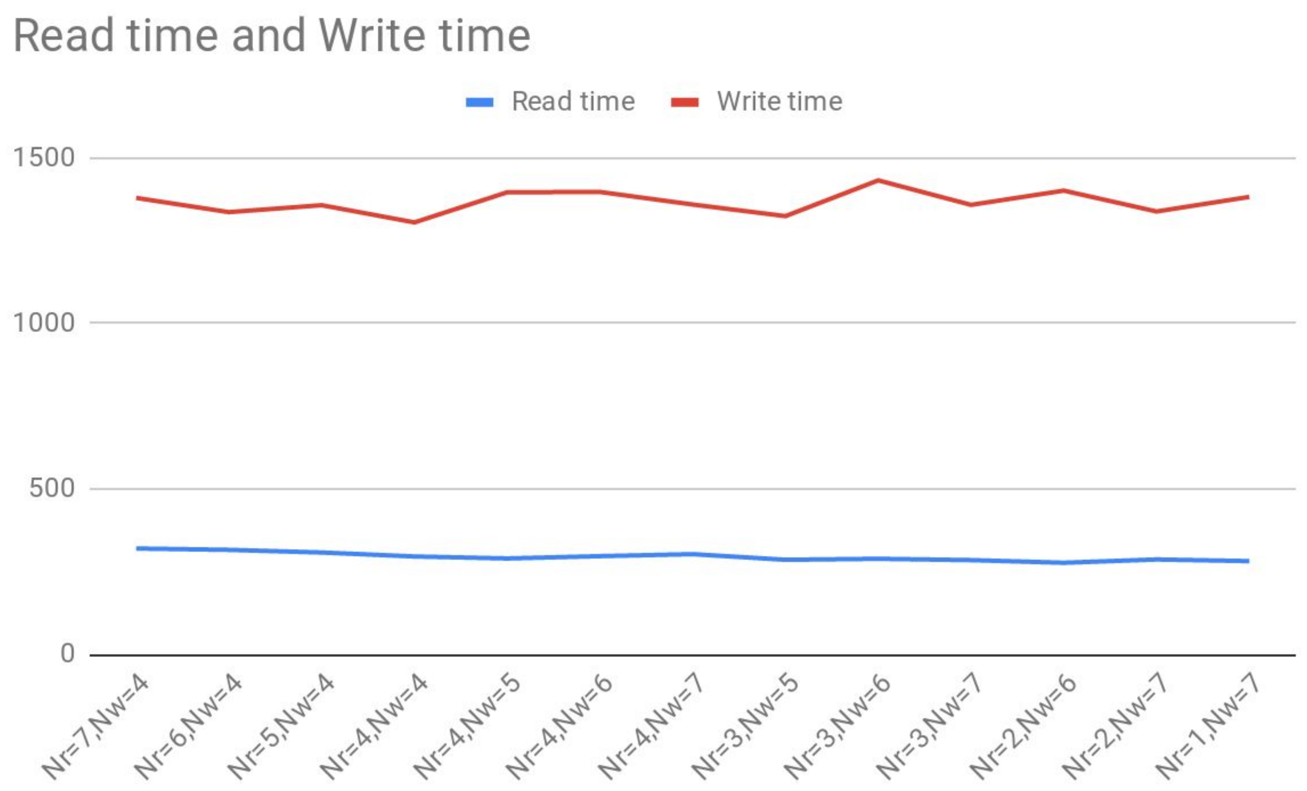
    Heavy read time taken = 287ms

    Heavy write time taken = 1338ms

13. Nr = 1, Nw = 7

    Heavy read time taken = 282ms

    Heavy write time taken = 1382ms



6. References

https://itss.d.umn.edu/services/file-storage

https://www.geeksforgeeks.org/java/

https://thrift.apache.org/tutorial/java

http://www-users.cselabs.umn.edu/classes/Spring-2019/csci5105/pas/pa3/pa3.pdf