

IT559 - Distributed Systems

Remote File Operation

Project Documentation

Group 2

Course instructor: Prof. P S Kalyan Sasidhar



April 20, 2024

Group Members

Member Name	Email
UPADHYAY MEET SHAILESHBHAI	202101042@daiict.ac.in
SABVA JAY DILIPBHAI	202101224@daiict.ac.in
THAKKAR MAULIK MUKESHBHAI	202101415@daiict.ac.in

Code - (Github): https://github.com/JaySabva/IT559-G2_Remote_File_Operation

Contents

1	Problem Statement	3
2	System Overview	3
3	Implementation Reference	3
4	Implementation	4
4.1	Write Operation	5
4.2	Read Operation	6
4.3	Functions	6
4.4	Results	7
5	Conclusions	9

1 Problem Statement

The Project aims to develop a system for remote file operations, allowing clients to perform read and write operations on the files stored on the remote servers.

2 System Overview

The system consists of three main components:

- Master Server:** Coordinates file operations, manage locks, and maintain metadata about servers and files stored.
- File Server (Primary):** Handles read and write operations for the primary copy of files.
- File Server (Backup):** Stores backup copies of files and synchronizes with the primary server to ensure data integrity.

3 Implementation Reference

- Our system implementation draws from the primary-backup protocol outlined in "Distributed Systems: Principles and Paradigms" by Andrew S. Tanenbaum and Maarten Van Steen.
- In this protocol, all write operations are forwarded to a fixed primary server, while read operations are executed from the backup servers.
- Primary-backup protocols ensure sequential consistency, as the primary orders all writes in a globally unique time order. Processes observe write operations in the same order, regardless of the backup server used for read operations.
- There are mainly two types of protocols named blocking and non-blocking. In blocking protocol, Other clients can not perform write operations until all the backup servers are update while in non-blocking protocol, multiple clients can perform write request and backup servers maintain sequential consistency in their updates.

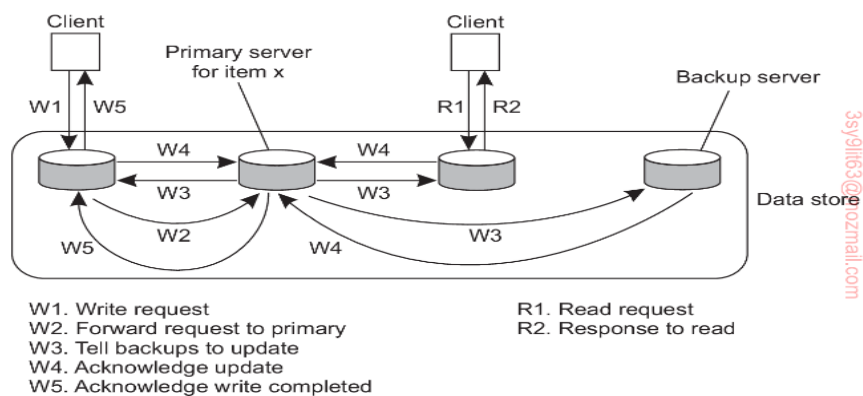


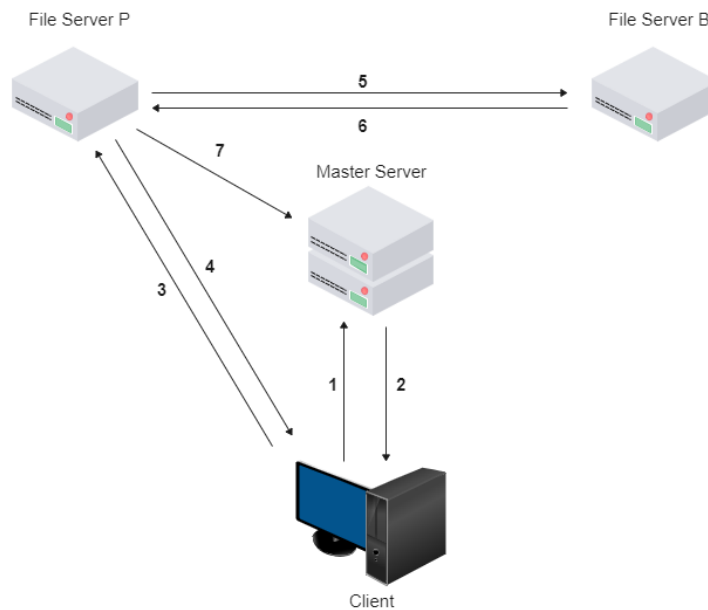
Figure 7.27: The principle of a primary-backup protocol.

4 Implementation

The implementation is based on the concept of RPC(Remote Procedure Call) for communication between different components of the system. Our system contains three major parts including a master server, primary server and a backup Server.

- a) Master server: The master server works as the central coordinating entity in our system. Its functionality includes lock management and metadata management.
 - Metadata management: It stores the information about file locations, including primary and backup servers. When client want to perform any operation on a file, it will contact master server for metadata.
 - Lock management: It handles request to lock and unlock files. When a client want to write on a specific file, master will allow the client to write on that file by locking the file exclusively for the client.
- b) File Server (Primary): When a new file is created, master server randomly allocate a server as the file's primary server. Every write request on that file will be handled by the primary server. Primary server sends updates to all the backup servers after performing the write operation. It also manages a priority queue (min heap) for the unsuccessful write operations on the backup servers.
- c) File Server (Backup): Every read operation on a file is handled by backup servers. If a backup server fails, then primary server will store the updates to its priority queue and when the backup server becomes available, it will ask for its updates from all the servers. Since we are maintaining priority queue (min heap) based on the timestamp of operations, all the updates will maintain sequential consistency.

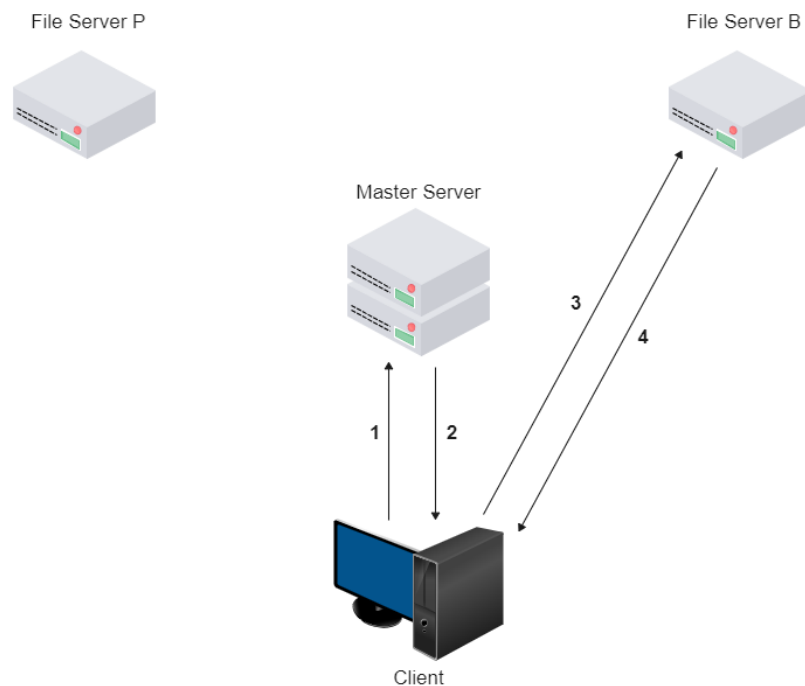
4.1 Write Operation



Above image shows how a write request made by client propagates in system.

1. First, client sends the write request to the master server with filename. For clarification, let us assume client **A** sends the write request to the master server with file name **hello.txt**. Master server checks whether **hello.txt** exist or not using metadata. If it doesn't exist, then the master server randomly chooses a file server as the primary server for **hello.txt** and it saves this information in the metadata. Master server also apply lock on **hello.txt** so the other clients can not perform write operation on the file before client **A** completes its write operation.
2. Master server provides the address and port number of the primary server of **hello.txt**.
3. Client **A** connects with the primary server using **xmlrpc** library. Client **A** provides data to be written and sends a write request to the primary server,
4. Primary server writes the data in its own local copy and return the success response to client **A**. Note that it is a non-blocking protocol.
5. After responding to client **A**, primary server sends the updates to the backup servers and backup servers writes those updates in their local copy of the file **hello.txt**. If a backup server is down then the operation along with the timestamp will be stored in the priority queue (min heap) corresponding to that backup server.
6. After completing write operation, backup servers send the success response.
7. Primary server creates the list of backup server which responded success and it send that list to the Master server, and master server add those server for read operation on file **hello.txt**.

4.2 Read Operation



Above image shows how a read request for the file **hello.txt** made by client **B** propagates in system.

1. Client **B** send the read request to the master server with filename **hello.txt**. Master server checks whether **hello.txt** exist or not using the metadata.
2. If the file exists in the metadata, then the master server sends address and port number of the backup servers with success response otherwise it responds as file not found.
3. Client **B** connect with the backup server using **xmlrpc** library and sends the read request to the backup server.
4. Backup server reads the data of **hello.txt** from its own local copy and sends the data as the response to the read request.

4.3 Functions

a) Master Server:

1. `write(filename)`: Initiates a write operation on the specified file by acquiring a lock and obtaining server information.
2. `lock(filename)`: Attempts to lock the specified file for writing, ensuring exclusive access; returns lock status and server information.
3. `unlock(filename)`: Releases the lock on the specified file, allowing other processes to write to it.

4. read(filename): Retrieves the list of backup servers associated with the specified file for read operations.
5. send_backup_servers(backup): Updates the backup_servers dictionary with information about backup servers for each file.

b) File Server:

1. write(filename, data, primary, flag, timestamp=None): Writes data to a file, either in append or write mode, and initiates backup if it's a primary server.
2. send_to_backups(filename, data, mode, timestamp): Sends data to backup servers asynchronously.
3. read(filename): Reads data from a specified file.
4. getMyUpdate(): Retrieves updates from other servers and writes them to local files. (when server starts this function is runned first)
5. sendUpdate(addr, port): Sends updates to other servers.

c) Client:

1. write_file(filename, mode): Initiates a write operation to the specified filename. It communicates with the master server to acquire the necessary lock and coordinates with the appropriate backup server to write the data.
2. read_file(filename): Initiates a read operation from the specified filename. It communicates with the master server to obtain information about the backup servers holding the file's data and retrieves the data from one of the backup servers.

4.4 Results

```

PROBLEMS OUTPUT TERMINAL PORTS SEARCH ERROR DEBUG CONSOLE
Select an option:
1. Write to a file
2. Read from a file
3. Quit
Enter your choice (1/2/3): 1
Enter the filename to write: Hello.txt
Enter 'w' for write or 'a' for append: w
Enter the data to write to the file: first
Data written to Hello.txt

Select an option:
1. Write to a file
2. Read from a file
3. Quit
Enter your choice (1/2/3): 1
Enter the filename to write: Hello.txt
Enter 'w' for write or 'a' for append: a
Enter the data to write to the file: second
Data written to Hello.txt

Select an option:
1. Write to a file
2. Read from a file
3. Quit
Enter your choice (1/2/3): 1
Enter the filename to write: Hello.txt
Enter 'w' for write or 'a' for append: a
Enter the data to write to the file: third
Data written to Hello.txt

Select an option:
1. Write to a file
2. Read from a file
3. Quit
Enter your choice (1/2/3): 2
Enter the filename to read: Hello.txt
Data in Hello.txt:
first
second
third

Select an option:
1. Write to a file
2. Read from a file
3. Quit
Enter your choice (1/2/3): 3
Exiting the program.
msm1@thuhkgshu1ike-MacBook-Air: ~ %

```

Figure 1: The figure shows various input provided by client to the server running on a remote system.

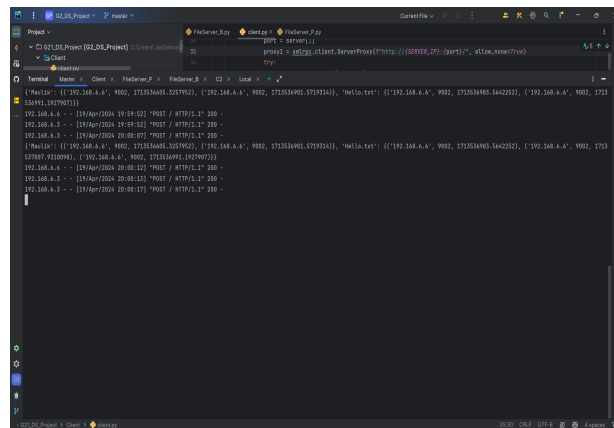


Figure 2: The figure shows the logs of master server.

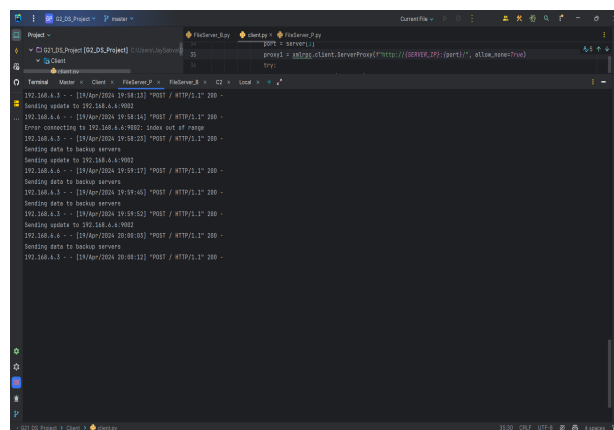


Figure 3: The figure shows the logs of primary server.

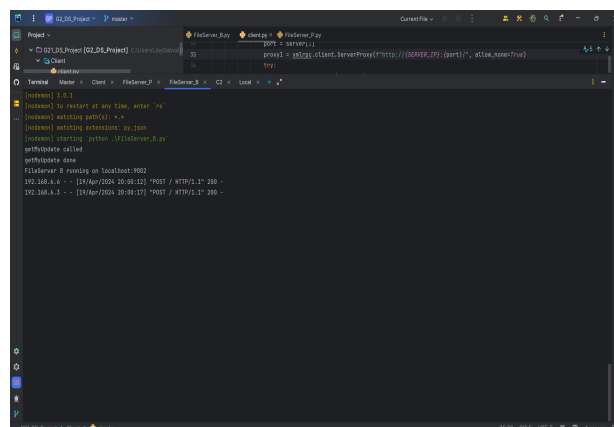
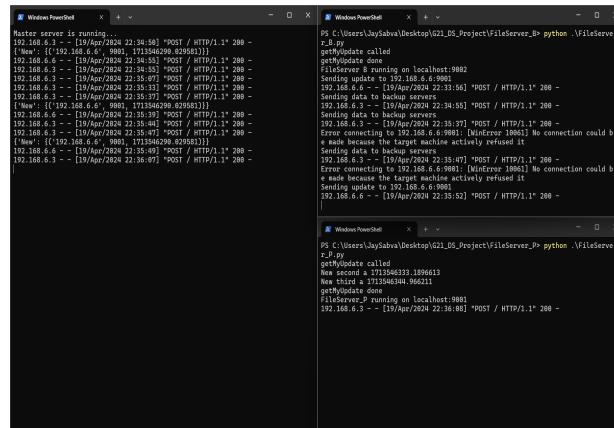


Figure 4: The figure shows the logs of backup server.



```

Master server is running...
192.168.6.3 - [19/Apr/2024 22:34:54] "POST / HTTP/1.1" 200 -
[New: [192.168.6.6, 9880, 1713846298.829583]]
192.168.6.6 - [19/Apr/2024 22:34:55] "POST / HTTP/1.1" 200 -
192.168.6.3 - [19/Apr/2024 22:34:55] "POST / HTTP/1.1" 200 -
192.168.6.3 - [19/Apr/2024 22:35:07] "POST / HTTP/1.1" 200 -
192.168.6.3 - [19/Apr/2024 22:35:13] "POST / HTTP/1.1" 200 -
192.168.6.3 - [19/Apr/2024 22:35:17] "POST / HTTP/1.1" 200 -
[New: [192.168.6.6, 9880, 1713846298.829583]]
192.168.6.6 - [19/Apr/2024 22:35:19] "POST / HTTP/1.1" 200 -
192.168.6.3 - [19/Apr/2024 22:35:19] "POST / HTTP/1.1" 200 -
192.168.6.3 - [19/Apr/2024 22:35:40] "POST / HTTP/1.1" 200 -
192.168.6.3 - [19/Apr/2024 22:35:47] "POST / HTTP/1.1" 200 -
[New: [192.168.6.6, 9880, 1713846298.829583]]
192.168.6.6 - [19/Apr/2024 22:35:48] "POST / HTTP/1.1" 200 -
192.168.6.3 - [19/Apr/2024 22:36:07] "POST / HTTP/1.1" 200 -

PS C:\Users\JaySabra\Desktop\G21_05_Project\FileServer_P> python .\FileServer
z_b.py
getUpdate called
FileServer_S running on localhost:9882
Sending update to 192.168.6.6:9880
192.168.6.6 - [19/Apr/2024 22:35:51] "POST / HTTP/1.1" 200 -
Sending data to backup server
192.168.6.3 - [19/Apr/2024 22:35:51] "POST / HTTP/1.1" 200 -
Sending data to backup server
192.168.6.3 - [19/Apr/2024 22:35:57] "POST / HTTP/1.1" 200 -
Error connecting to 192.168.6.6:9881. Unidrive 198613 No connection could b
e made because the target machine actively refused it
192.168.6.3 - [19/Apr/2024 22:35:47] "POST / HTTP/1.1" 200 -
Error connecting to 192.168.6.6:9881. Unidrive 198612 No connection could b
e made because the target machine actively refused it
Sending update to 192.168.6.6:9880
192.168.6.6 - [19/Apr/2024 22:35:52] "POST / HTTP/1.1" 200 -

PS C:\Users\JaySabra\Desktop\G21_05_Project\FileServer_P> python .\FileServe
r_z_b.py
getUpdate called
New record a 1713846333.1896613
New third a 1713846344.966211
getUpdate done
FileServer_P running on localhost:9881
192.168.6.3 - [19/Apr/2024 22:36:08] "POST / HTTP/1.1" 200 -

```

Figure 5: The figure shows the updates done by backup server.

5 Conclusions

In conclusion, we have successfully developed a system supporting remote file operations utilizing the primary backup protocol. Through the implementation of RPC protocol and the integration of a priority queue based on timestamps, we have successfully achieved sequential consistency in file operations across servers.

Our system offers reliable and efficient remote file operations, enabling users to perform tasks seamlessly across different nodes without compromising data integrity or consistency. By leveraging the primary backup protocol, we ensure that any changes made to files are synchronized across the network, providing consistency and fault tolerance.

For the future enhancements, we can add a functionality which can dynamically change the primary server if current primary server fails. It will increase the fault tolerance of existing system.