PCS PROJECT DESIGN DOC

SECURED DISTRIBUTED FILE SYSTEM

Group 5 - Authors:

Pooja Gopu (sr21814@umbc.edu)
Rithin Nandikonda (rithinn1@umbc.edu)
Akshith Reddy Chada (rh34539@umbc.edu)
Ray Karyshyn (raykaryshyn@umbc.edu)

Design Assumptions:

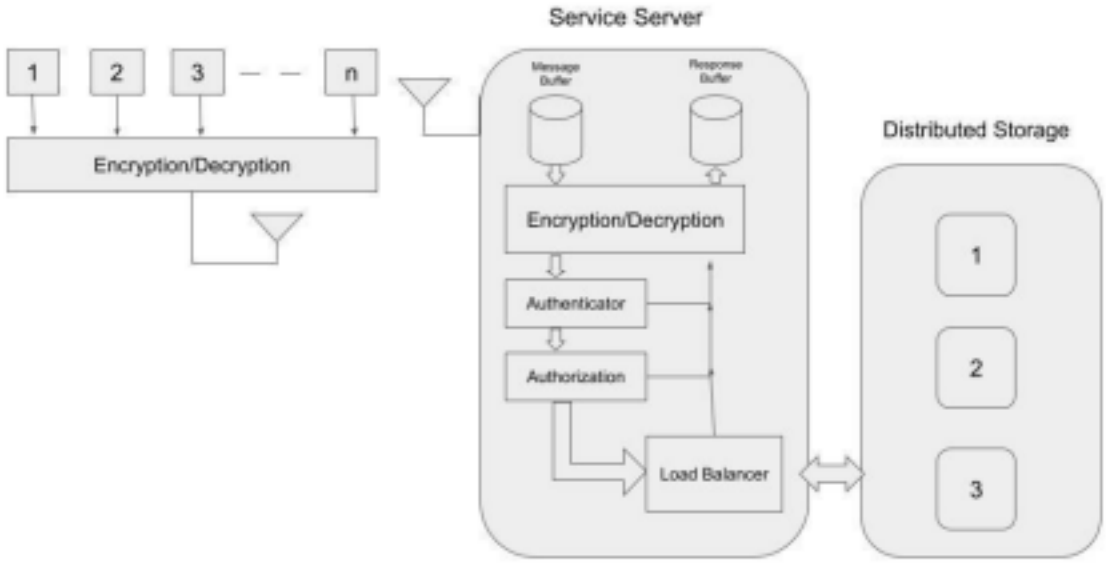
The aim of the project is to develop a secured distributed file system. The users will be able to connect to the file server and access saved information. To achieve this, we are making the following design assumptions.

- We are defining a fixed file size for all the files.
- The server can read, write and delete all files on its hard disk.
- Multiple reads can happen at same time on a single same file
- Only single write can happen on a given single file; locks will be applied when performing write/update.
- We assume that there is enough memory in the server for all requests that are being processed.
- The system will provide authentication and authorization to users/clients which will prevent unauthorized users from accessing the information.
- For each file in the system, there will be a set of reader and writer authority.

Objectives:

The objective of the project is to build a distributed file system with security. This system consists of an array of nodes. In this system users can read, write and delete any file depending on the user's access level, that is reader or writer. We will use a robust replication scheme to have data consistency across the system. Even if one server in the cluster has a corrupt file the system will be able to retrieve the data from other reader or writer nodes. The file system will be able to handle 100k concurrent requests. Client view will be like accessing its local file system, need not to worry about system failure etc.

UML diagram:



Functionality:

- The end user can do the following operations: Read, Write, Delete.
- The primary server maintains all the information about the replicated server records, user access for operations. It also performs encryption, decryption and user auth. ● The client requests the primary server using RMI, similarly the primary server does the replication.
- To maintain consistency across the system, the data is not reflected until the server is done replicating across all servers.
- To prevent multiple users from making changes to the same files i.e. write and delete, we are going to use concurrent collections and logs such as mutexes on the object the operation is being performed on.
- All storage servers will be in sync using P2P communication
- Consider a case when a particular storage server goes down and reboots, it must get back in sync with other servers. For this we will use P2P communication to update the data in the crashed server from the server which has the latest data with encryption/decryption involved.

Request Flow:

- When the client makes a request, the request goes to the primary server where the user or client is authenticated and next, we use a thread safe collection such as concurrent HashMap for the file's directory and user access levels.
- Here we use locks on the object which is being modified to prevent multiple users from performing operations simultaneously.
- Then the file is decrypted and operations (Read, Write, Delete) are performed.
- After the operations are performed, replication is done to all the other servers after which the transaction is committed and the changes in the data reflect for all other users.

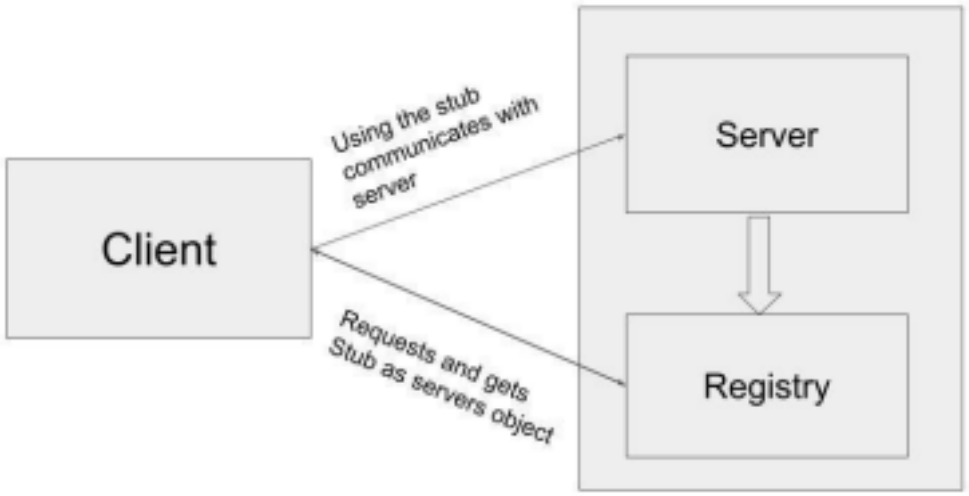
RMI: Remote method invocation

- It's an API which can serve the distributed systems as a remote calling mechanism is at its core.

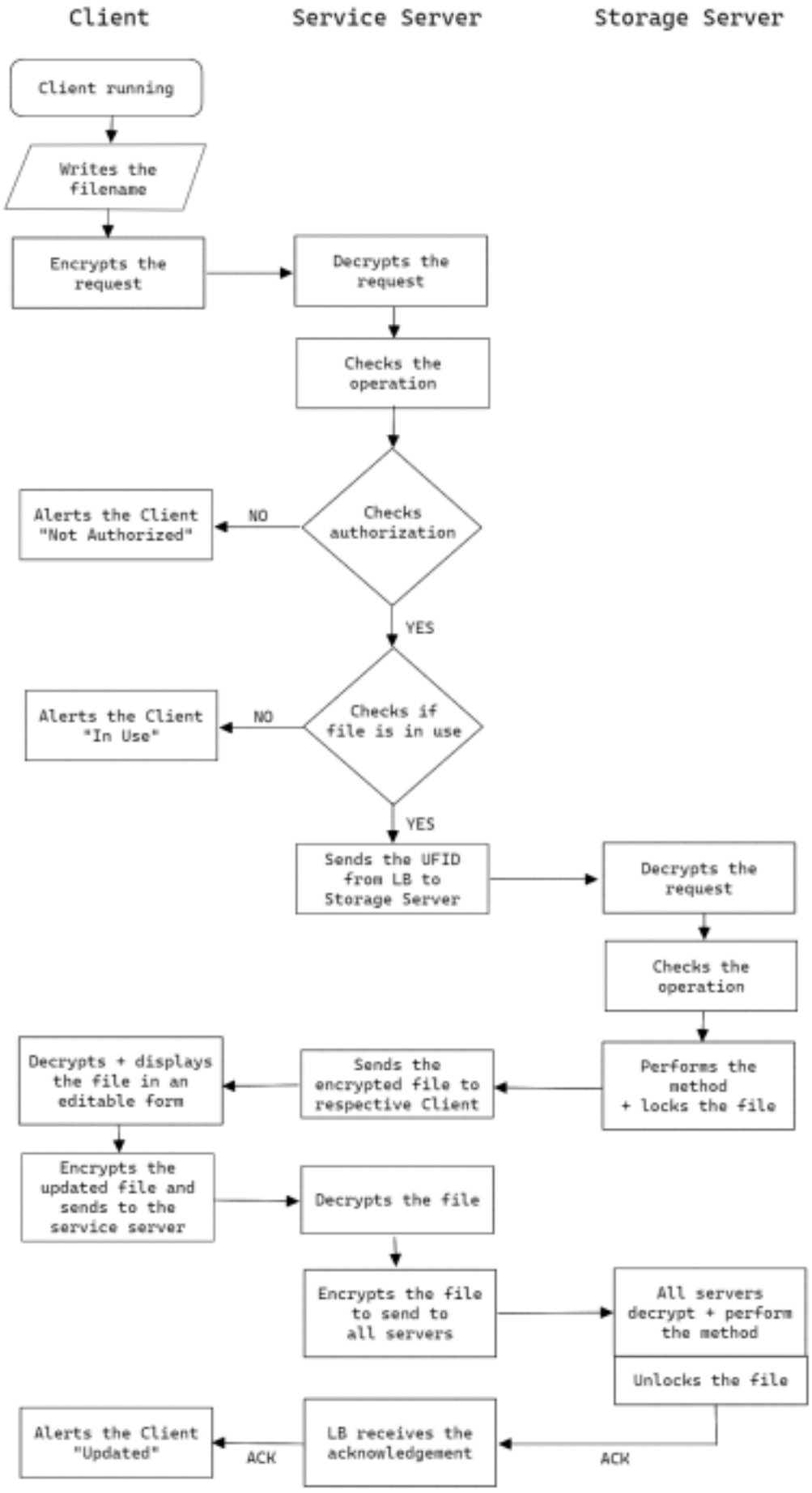
● Allows an object to call a method on an object running in a remote machine. ● RMI is a higher-level abstraction with use of TCP communication and doesn't need to worry about multithreading.

- Basically, the client has a remote object of the server which will be used to invoke a method on an object on the remote server to get the processing done on the server and return the results.

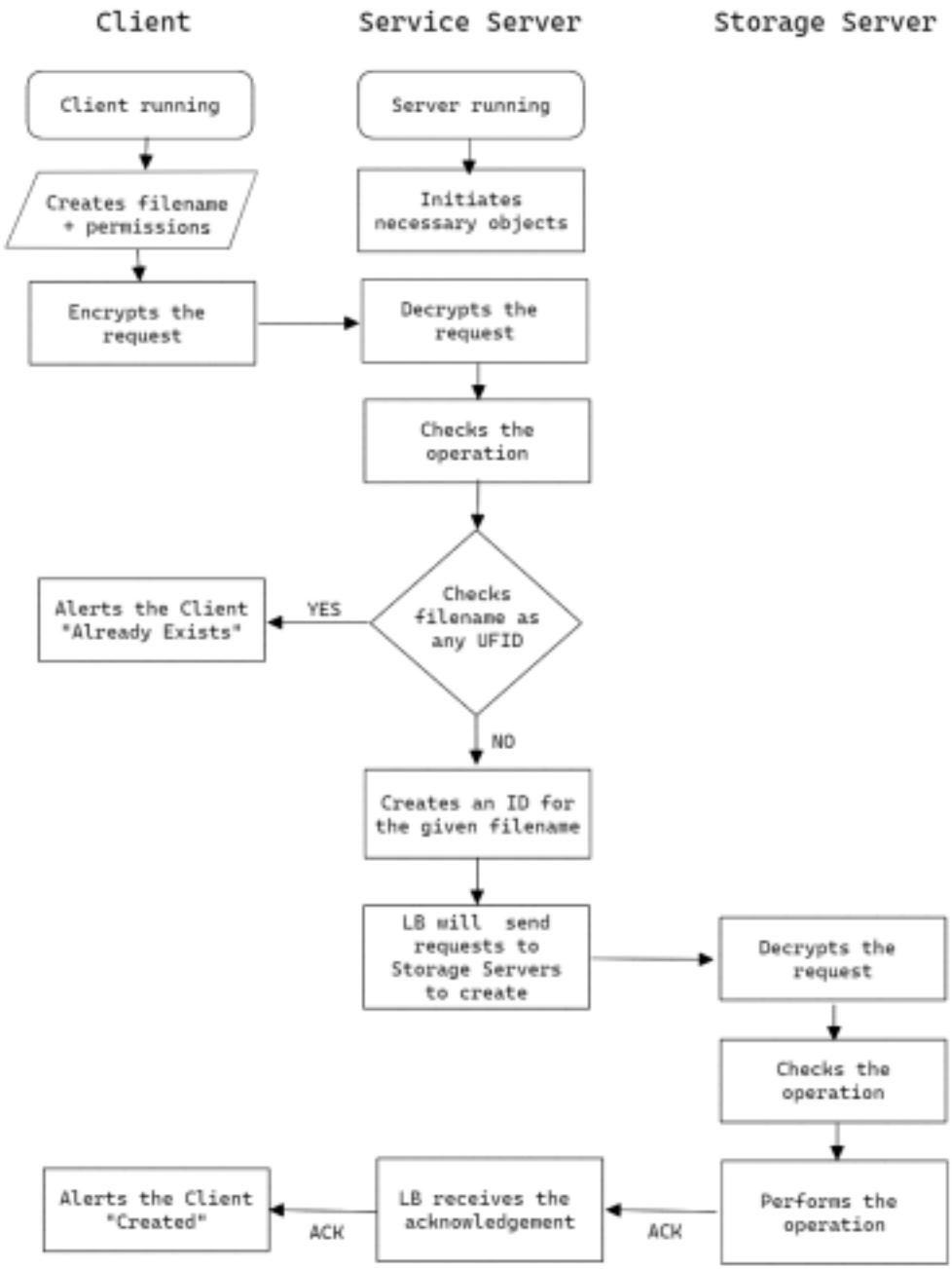
● Server loads the objects into the registry (lookup table). Client requests the stub instance from the server's registry and that's the gateway for communication on client side. ● Therefore, using the stub client invokes objects on the server.

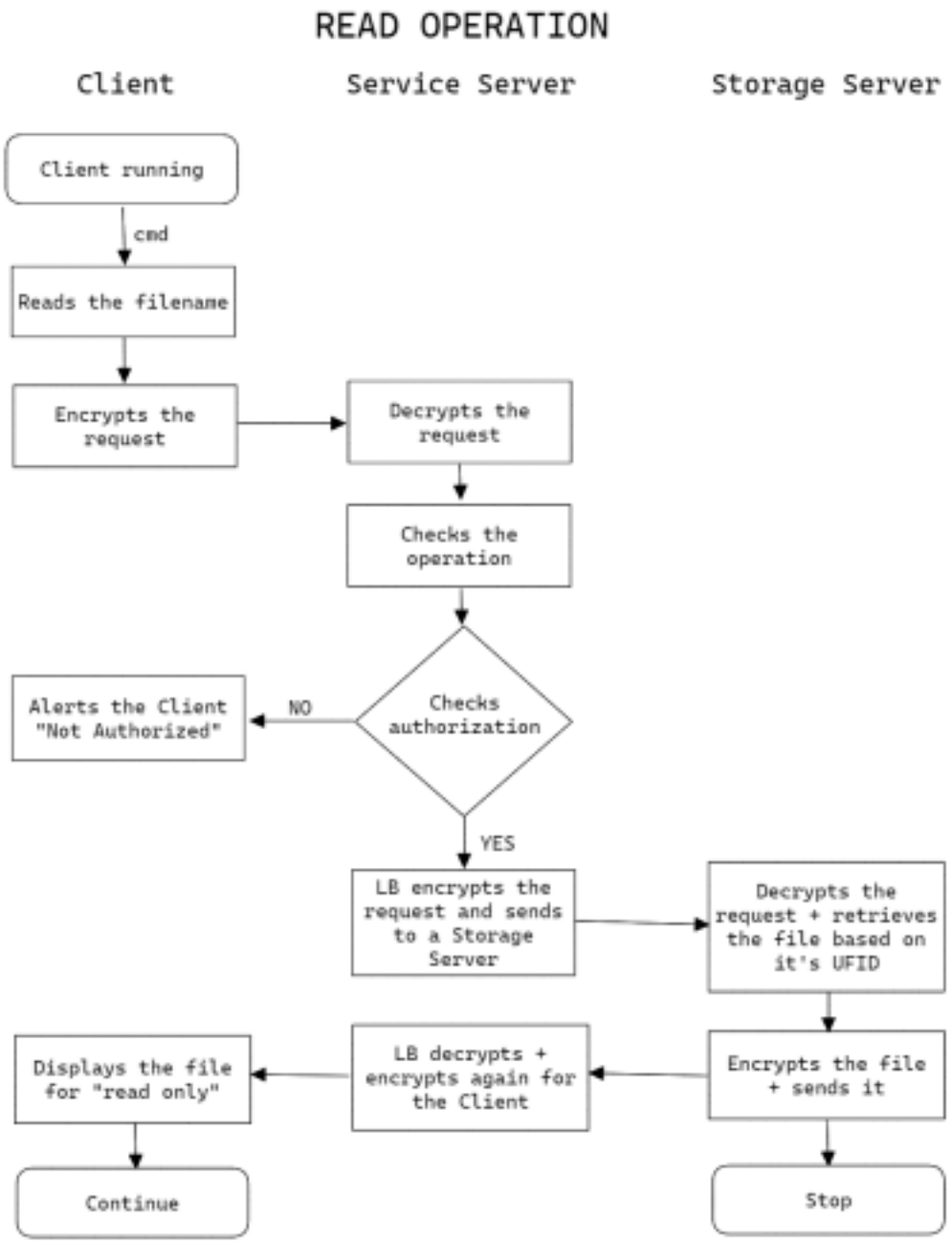


WRITE OPERATION



CREATE OPERATION





Libraries and Code:

- We are planning to develop this project in Java.
- We will be using virtual servers and RPC protocol instead of traditional web APIs for primary and replica servers.
- We will be running all servers as virtual machines.
- We will also develop code for simulating client-server testing.
- We will use RMI API (or similar) for RPC calls.
- We will use basic authentication or oAuth 2.0 for user authentication and inbuilt encryption libraries in Java for file encryption.

References:

https://www.geeksforgeeks.org/file-service-architecture-in-distributed-system/

https://www.oracle.com/technical-resources/articles/javase/rmi-corba.html

https://www.cse.scu.edu/~m1wang/projects/DFS_simulation_14m.pdf

https://homes.cs.aau.dk/~bnielsen/DS-E06/Java-RMI-Tutorial/