Project Initial Design

CS4262 - Distributed Systems

Team Members:-

Shanmugavadivel Gopinath	190199A
Sivaparan Sivakajan	190597R
Tharsha Sivapalarajah	190599B

Our Secure and Trusted Distributed File Storage System allows data to be scattered across multiple storage devices on a network. This enables file retrieval whenever needed. This distributed approach offers several advantages:

1. Enhanced Scalability
2. Improved Fault Tolerance
3. Increased Security

Here's a breakdown of the workflows within the Secure and Trusted Distributed File Storage System:

1. Random Distribution of File Chunks

a. Chunking the File:

i. Read the file into memory.

ii. The master node will split the file into fixed-size chunks (C) of data. We have planned to use a Fixed-Size chunking algorithm for this purpose. Fixed-Size Chunking (FSC) is a Deduplication algorithm that breaks the data into fixed-size chunks or blocks from the beginning of the file.

b. Random Distribution:

i. Assign each chunk to a random node in the cluster. We have planned to utilize a dynamic load-balancing algorithm to evenly distribute the chunks across the cluster. This algorithm monitors slave node load and redistributes file chunks accordingly ensuring optimal resource utilization and preventing overloading of individual slave nodes.

ii. Ensure redundancy by replicating each chunk across multiple slave nodes for fault tolerance. The degree of replication can be predefined. I.e we will use a replication factor (e.g., 3) to determine the number of copies for each chunk.

2. Ensuring Trustworthiness with Merkle Tree

a. **Generate Merkle Tree**:

   For each uploaded file, a Merkle Tree will be generated based on the chunks of the file. During this process, the hash value of each chunk will be calculated, forming the leaf nodes of the Merkle Tree. Subsequently, adjacent leaf nodes' hash values will be combined to construct parent nodes iteratively until reaching the root node, which encapsulates the hash value for the entire file.

b. **Store Merkle Tree and Metadata**:

   The master node has to keep the location and information regarding each slave node and the chunks they contain. The metadata of each file, including the file name, size, date of storage, and type, along with the root hash value of the Merkle Tree, are intended to be stored in the master node. To avoid single point of failure, the Secondary master node is maintained and it is kept consistent by having frequent communication with the master node.

3. User Interface

We have planned to implement a CLI-based user interface to connect users with our system. The interface will have the following functionalities.

a. **List Files**: Display the list of files with their metadata attributes - file name, size, date of storage, and type.

b. **Search Files**: Search functionality allows users to search for files based on metadata attributes.

c. **View Metadata**: Display metadata details for a specifically selected file.

d. **Verify File Integrity:** Verification functionally allows the user to check whether the file has been tampered with or not.

   During this process, the root hash of the corresponding Merkle Tree will be retrieved from the distributed database. Subsequently, the Merkle Tree will be recomputed based on the file chunks stored across the cluster. Upon completion, the computed root hash can be compared with the stored root hash to confirm the file's integrity.

e. **Download File**: Downloading functionality allows users to download needed existing files. During this process, the file chunks associated with the specific file will be retrieved from the cluster nodes and will be assembled into a single file.

f. **Upload File:** Uploading functionality allows users to upload new files to the system. During this process, the specific file will be segmented into several equal-sized chunks and will be stored on the cluster nodes.