Secure File Storage and Retrieval System Report

Architecture and Functionality

The Secure File Storage and Retrieval System is designed to provide a robust and secure method for storing and retrieving files using a client-server architecture. The system leverages Merkle tree verification to ensure the integrity of the files. The client can upload files to the server, delete local copies, and later download any specific file with a guarantee of its integrity.

Architecture

The system is composed of two main components:

1. **Client**: Responsible for uploading, updating, and downloading files. It also handles the generation and verification of Merkle tree proofs to ensure file integrity.
2. **Server**: Manages the storage of files and their corresponding Merkle tree nodes. It also handles requests from the client for file uploads, updates, and downloads.

Functionality

- **Upload**: The client uploads files to the server, generating a Merkle tree to ensure the integrity of the files. The root hash of the Merkle tree is stored locally on the client's machine.
- **Update**: The client can add new files to the existing set without deleting the existing files. This is achieved by updating the Merkle tree and sending the new root hash to the server.
- **Download**: The client can download a specific file by providing the index of the file. The server sends the file along with the Merkle proof, which the client verifies using the locally stored root hash.

TODOs

Throughout the code, I have placed TODO comments indicating areas where improvements can be made. These comments serve as markers for potential enhancements and optimizations.

Challenge

The main challenge was to create a system that could support multiple users. To achieve this, we used the root hash as a unique client identifier. This approach allowed the system to generate blocks (nodes) for a set of files and add new files to existing uploads (using the update operation) without deleting existing files.

To make this work, we store the root hash on the user's local machine and send it along with the files during uploads and updates.

Key Issues

1. **Proof Generation**: Generating the Merkle proof for file integrity verification was challenging. Ensuring that the proof was accurate and could be verified correctly required careful implementation.
2. **Proof Verification**: Verifying the Merkle proof, especially determining the correct sides (left or right) of the nodes, was complex. This required creating functions to display the entire tree and all proofs for visual verification and debugging.
3. **Local File Hashing**: To ensure integrity, we needed to generate the hash of the local file and compare it with the hash stored in the Merkle tree.
4. **Node Verification**: Creating functions to display the entire tree and all proofs helped in visually verifying and understanding where errors were occurring.
5. **Client Tests**: There were issues with the client tests, particularly with handling flags.
6. **Mocking ScyllaDB**: Mocking the ScyllaDB database for testing purposes was challenging.
7. **Multi-User Support**: Ensuring that the system could handle multiple users simultaneously required careful consideration of user-specific data storage and retrieval.
8. **LevelDB Iterator**: Handling the LevelDB iterator and managing buffers in the iterators was complex.
9. **File Deletion**: Ensuring that files were deleted correctly when using the upload operation with new files or after an update.

Improvements

1. **Recursive vs. Iterative**: Consider using an iterative approach instead of a recursive one for certain operations to improve performance and reduce stack overflow risks.
2. **HTTPS**: Implement HTTPS to ensure secure communication between the client and server.
3. **Context Propagation**: Use and propagate context for observability, timeout, and cancellation. This will improve the system's robustness and manageability.
4. Observability: Implement observability features to monitor the system's performance and detect issues in real-time.
5. **Kubernetes, ArgoCD, and Helm**: Use Kubernetes for container orchestration, ArgoCD for continuous delivery, and Helm for package management to improve deployment and scalability.
6. **Bulkhead Pattern**: Implement the Bulkhead pattern to isolate different parts of the system and prevent failures from cascading.
7. **Streams**: Use streams for efficient data processing and transfer.

Conclusion

The Secure File Storage and Retrieval System successfully provides a secure and reliable method for storing and retrieving files using a client-server architecture with Merkle tree verification. While there were several challenges, particularly with multi-user support and proof verification, the system is functional and can be further improved with the suggested enhancements. The use of TODO comments in the code highlights areas for potential improvement and optimization.