1 Introduction

This project focuses on designing a distributed file system inspired by the Google File System (GFS). The system emphasizes exactly-once semantics for append operations while ensuring scalability, high availability, and fault tolerance.

Key Features:

- **Exactly-Once Append Semantics**: Ensures data is appended exactly once, even during retries or failures.
- **Scalable and Fault-Tolerant Design**: Incorporates versioning, replication, and lightweight communication protocols.

2 System Design

2.1 Architecture Overview

The system is designed with distinct layers:

- **Master Server Layer**: Oversees global metadata, coordinates operations, and ensures atomicity.
- **Chunkserver Layer**: Stores data chunks with replication for reliability.
- **Client Layer**: Interfaces with the user and facilitates file operations such as CREATE, READ, APPEND, and DELETE.

2.2 Component Responsibilities

Master Server:

- Maintains metadata, including file-to-chunk mappings, chunk locations, and versioning.
- Coordinates operations using a two-phase commit (2PC) protocol to ensure atomicity.
- Handles chunk CREATE, DELETE, APPEND, directly with chunkserver and updated metadata updates.
- Helps Client to get Location of chunkserver in READ.

Chunkserver:

- Performs Real operation on Disk.
- Stores data in a flat namespace (`{filename}_{chunk_number}_{version}`).
- Maintains logs for prepared transactions and metadata for tracking completed operations.

- Ensures replica consistency during failures or retries.

Client:

- CLI based interface enables the user to perform required actions.
- Requests metadata and coordinates operations via the master server.
- Communicates directly with chunkservers for data retrieval.

2.3 Communication Protocol

Lightweight protocols like HTTPs are used for inter-component communication:

- **Client Master**: Client requests for CREATE, READ, DELETE, and APPEND operations. Master give Acknowledgement of Operation in READ it sends the location of chunk-servers
- **Master Chunkserver**: Coordination for chunk Creation, deletion, and Chunk updates.
- **Client Chunkserver**: Coordination for data retrival in READ.

3 Exactly-Once Semantic Append

3.1 Challenges and Solutions

Achieving exactly-once semantics for append operations in a distributed file system involves addressing several key challenges. Below, we outline these challenges and their corresponding solutions:

1. **Retries During Failures**
   Challenge: When an operation fails and is retried, it may result in duplicate appends
   Solution: Implementing *versioning* ensures idempotency by tracking and validating the version number of each chunk. This guarantees that retries do not create duplicate entries.

2. **Partial Appends**
   Challenge: Failures during the execution of an append operation may leave the system in an inconsistent state, where some chunkservers have committed the operation while others have not.
   Solution: Using the *Two-Phase Commit (2PC)* protocol ensures atomicity by requiring all chunkservers to acknowledge readiness before finalizing the operation.

3.2 Design Principles

- Each chunk is assigned a version number, incremented with updates to ensure idempotency.

Two-Phase Commit (2PC):

- Guarantees atomicity and consistency across chunkservers.
- **Prepare Phase**: Verifies readiness.
- **Commit Phase**: Finalizes the operation.

Replica Synchronization:

- Ensures consistent state among replicas through versioning.

Client Retry with Idempotency:

- Transaction IDs are used to prevent duplicate operations.

3.3 2PC Workflow

1. **Client Request**: Initiates the append operation via the master.
2. **Master Coordination**: Identifies chunkservers, locks chunks, and distributes data.
3. **Prepare Phase**: Chunkservers validate requests, lock resources, and send acknowledgments.
4. **Commit Phase**: Master finalizes the operation upon receiving all acknowledgments.
5. **Failure Handling**: Master aborts or retries the operation as needed.
6. **Client Notification**: Master informs the client of the operation's result.

4 Robustness in Failure Scenarios

4.1 Common Failures and Solutions

1. **Network Partition**:
   - The master waits for acknowledgments from all chunkservers or aborts the operation on timeout.
   - Upon recovery, chunkservers send their metadata back to the master to synchronize and reconcile.
2. **Chunkserver Crash**:

- If a chunkserver fails during an operation, it does not persist logs to disk. However, if sufficient replicas are available, the operation can still be performed (e.g., the read or write operation can be completed from the surviving chunkservers).
- Once the chunkserver recovers, it resynchronizes with the master and other chunkservers by re-communicating its metadata.

3. **Master Server Crash**:
   - The master does not persist logs. Upon recovery, all chunkservers resend their metadata to the master to reestablish consistency and ensure that all operations are properly recorded.

4. **Duplicate Client Requests**:
   - The master and chunkservers use transaction IDs and versioning to detect and prevent duplicates. If a client retries an operation due to a timeout, the system ensures the request is not executed again, avoiding duplication.

4.2 Guarantees

- **Atomicity**: All or none of the chunkservers commit the operation.
- **Consistency**: All replicas reflect the same state post-operation.
- **Idempotency**: Retries do not result in duplicate appends.

5 Future Scope

- **Arbitrary Write Operations**: Extend support for general write operations with consistency guarantees.
- **Dynamic Scaling**: Automate the addition of chunkservers and rebalance data dynamically.
- **Load Balancing**: Distribute file chunks to prevent hotspots.

6 Conclusion

This system integrates exactly-once append semantics using versioning, 2PC, and retry idempotency, ensuring a scalable and fault-tolerant distributed file system. With its extensible architecture and robust failure handling, it is well-suited for modern distributed storage needs, while offering opportunities for future enhancements.