Proposal for Implementing GFS with Exactly-Once RecordAppend Semantics

Team Members

1. Mitansh Kayathwal — 2021101026
2. Vineeth Bhat — 2021101103

1. Statement of the Problem

The **Google File System (GFS)** was designed to handle large-scale distributed data storage and processing, supporting multiple concurrent clients writing to large files. One of the key features of GFS is its **RecordAppend** operation, which allows multiple clients to append to a file. However, in the original GFS implementation, RecordAppend provides **at-least-once** guarantees, meaning that the system ensures the data is written at least once, but duplicates might be introduced.

The problem addressed in this project is the enhancement of GFS's RecordAppend semantics from **at-least-once** to **exactly-once**, where each append operation will be guaranteed to happen only once, even in the face of client failures, network partitioning, or chunk server crashes. Achieving exactly-once semantics in a distributed system is non-trivial and requires careful handling of:

- **Network failures**
- **Retries**
- **Consistency across replicas**

2. Important Papers/Materials

To inform the design and implementation of the system, the following materials are crucial:

- **Google File System (GFS) Paper**: This foundational paper describes the architecture and operations of GFS. Understanding how GFS currently handles RecordAppend with at-least-once semantics is key to identifying where enhancements can be made for exactly-once guarantees.
    - **Reference**: *Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google File System." SOSP 2003.*
- **Paxos Made Simple (Lamport, 2001)**: This paper explains the Paxos consensus algorithm, which is a cornerstone for achieving fault-tolerant consensus across distributed systems. Paxos or similar consensus algorithms could be used to ensure exactly-once semantics across multiple chunk servers.

    - **Reference**: *Lamport, Leslie. "Paxos Made Simple." 2001.*
- **Viewstamped Replication Revisited (VRR, OSDI 2012)**: This paper discusses a replication protocol that guarantees consistency and fault tolerance. It may provide useful insights into managing consistency across replicas for exactly-once RecordAppend.
- **RAFT Consensus Algorithm (Diego Ongaro & John Ousterhout, 2014)**: RAFT simplifies the implementation of consensus, making it easier to implement exactly-once guarantees in distributed systems.
    - **Reference**: *Ongaro, Diego, and John Ousterhout. "In search of an understandable consensus algorithm." USENIX 2014.*

3. Scope of the Project

This project will involve the implementation of a simplified **Google File System (GFS)** from scratch, but with a key enhancement: modifying the **RecordAppend** operation to provide **exactly-once** semantics. This enhancement is useful in environments where data deduplication is critical, such as logging systems or transactional data stores, where data integrity is paramount.

Feature Overview

- **Exactly-Once RecordAppend Semantics**:
    - Ensures that each record is appended exactly once, preventing duplicates even in failure scenarios.

Use Cases

- **Distributed Logging**: Ensures that each log entry is written only once, preventing duplicate log messages that could lead to incorrect diagnostics.
- **Transactional Systems**: Guarantees exactly-once data writes in applications where consistency is critical, such as banking systems, inventory management, or billing systems.

4. Proposed Solution

The core challenge is ensuring that an append is executed exactly once even when failures occur. The following design principles will be followed:

Solution Design

1. **Client-Side Idempotence**:
    - Each RecordAppend request from a client will carry a **unique identifier (UUID)**. This ensures that the master and chunk servers can detect retries (from client failures or timeouts) and distinguish between new requests and retries.
2. **Master and Chunk Servers Coordination**:
    - **Master Server**: Tracks metadata and chunk placement while monitoring whether a particular UUID has already been processed to prevent

reprocessing requests.

   - **Primary Chunk Server**: Responsible for coordinating the append operation while keeping track of received UUIDs to reject duplicates.
   - **Replicas**: All replicas will use the UUID to identify append operations and prevent duplicate writes; coordination by the master ensures consistency.

3. **Consensus Mechanism (RAFT or Paxos)**:
   - To ensure that replicas remain consistent during failures, a consensus mechanism (such as RAFT or Paxos) will be implemented to ensure all chunk replicas agree on whether an append operation has been applied.
   - **Leader Election**: The primary chunk server acts as the leader coordinating append operations while replicas act as followers; consensus manages failover when a primary chunk server fails.
4. **Handling Failures and Retries**:
   - **Client Failure**: If a client fails while appending, UUIDs ensure that upon recovery, appends are not duplicated.
   - **Chunk Server Failure**: If a chunk server fails, the master can assign a new replica and synchronize state using consensus protocols to ensure appends are neither lost nor duplicated.

Technologies & Components

- **Programming Language**: Go (for concurrency and ease of network programming).
- **Consensus Algorithm**: RAFT or Paxos for ensuring consistency across chunk replicas.
- **gRPC**: For communication between master servers, chunk servers, and clients.
- **Persistent Storage**: Disk-based storage for state consistency.

Development Components

- **Client**: Sends RecordAppend requests uniquely identified by UUIDs.
- **Master Server**: Manages file metadata and tracks processed UUIDs.
- **Chunk Servers**: Store actual data chunks while applying exactly-once semantics for RecordAppend; participate in consensus for replication consistency.
- **Consensus Module**: Ensures agreement between chunk replicas preventing duplicate appends.

Dataset

A sample log file dataset or synthetic dataset will be used to simulate multiple clients appending data concurrently. This will allow testing of exactly-once semantics under real-world conditions.