GFS with Exactly-Once Semantic Append

Harsh Gupta - 2022101067, Ishan Gupta - 2022111007

Team 31

Introduction

Project Overview:

- Designing a distributed file system inspired by Google File System (GFS)
- Focus on exactly-once semantics for append operations
- Ensures scalability, high availability, and fault tolerance

Key Features:

- Exactly-Once Append Semantics
- Scalable and Fault-Tolerant Design

System Design Overview

Key Layers:

1. Master Server Layer: Global metadata management, operation coordination, atomicity
2. Chunkserver Layer: Stores data chunks with replication for reliability
3. Client Layer: User interface for file operations (CREATE, READ, APPEND, DELETE)

Component Responsibilities

Master Server:

- Manages metadata, coordinates operations using 2PC
- Ensures atomicity, handles chunk operations

Chunkserver:

- Performs disk operations, stores data, manages logs
- Maintains replica consistency

Client:

- CLI interface for interacting with the system
- Coordinates operations with master server and chunkservers

Communication Protocol

Protocols Used:

- TCP sockets for inter-component communication

Client to Master:

- Requests for file operations (CREATE, READ, DELETE, APPEND)

Master to Chunkserver:

- Coordination for chunk operations

Client to Chunkserver:

- Data retrieval for READ operations

Exactly-Once Semantic Append

Challenges:

- Retries During Failures
- Partial Appends

Solutions:

- Versioning for idempotency
- Two-Phase Commit (2PC) for atomicity
- Replica synchronization for consistency

Design Principles:

- Versioning for chunk updates
- 2PC for coordination across chunkservers
- Transaction IDs for client retries

Two-Phase Commit (2PC) Workflow

1. Client Request: Initiates append operation via master
2. Master Coordination: Identifies chunkservers and locks resources
3. Prepare Phase: Chunkservers validate and lock resources
4. Commit Phase: Master finalizes the operation
5. Failure Handling: Master aborts or retries operations as needed
6. Client Notification: Master informs the client of the operation's result

Robustness in Failure Scenarios

Common Failures and Solutions:

1. Network Partition: Master waits for acknowledgments or aborts
2. Chunkserver Crash: Operation can be completed using remaining replicas
3. Master Server Crash: Chunkservers synchronize on recovery
4. Duplicate Client Requests: Handled via transaction IDs and versioning

Guarantees:

1. Atomicity
2. Consistency
3. Idempotency

Future Scope

- Arbitrary Write Operations: Extend support for general write operations with consistency
- Dynamic Scaling: Automate chunkserver addition and data rebalancing
- Load Balancing: Prevent hotspots by distributing file chunks more evenly

Conclusion

Key Takeaways:

- Exactly-once semantics achieved through versioning, 2PC, and retry idempotency
- Scalable and fault-tolerant design
- Robust failure handling ensures system reliability
- Extensible for future enhancements