Distributed systems course project

EXACTLY ONE SEMANTICS IN RECORD APPEND IN GFS

TEAM 41

PRIET UKANI(2022111039), SAHIL PATEL(2022101046)

AGENDA OVERVIEW

01

PROJECT SCOPE

02

INTRODUCTION TO
DISTRIBUTED FILE SYSTEMS

03

GOOGLE FILE SYSTEM
ARCHITECTURE

04

SEMANTICS: AT-LEAST-ONCE
VS EXACTLY-ONCE

05

IMPLEMENTATION OF
EXACTLY-ONCE SEMANTICS

06

OPERATIONS
IMPLEMENTATION

07

RESULTS

08

LIMITATIONS AND
FUTURE WORK

PROJECT SCOPE

The scope of this project is to design and implement a distributed file system inspired by the Google File System (GFS) with a primary focus on achieving exactly-once append semantics.

01

FUNCTIONAL SCOPE

- Distributed file system with a client-server architecture
- operations such as CREATE, READ, APPEND, and DELETE.
- Provide exactly-once append semantics,

02

TECHNNICAL SCOPE

- Use master, chunkserver and client layer.
- Implementing 2PC protocol for operations
- Failure Handling

03

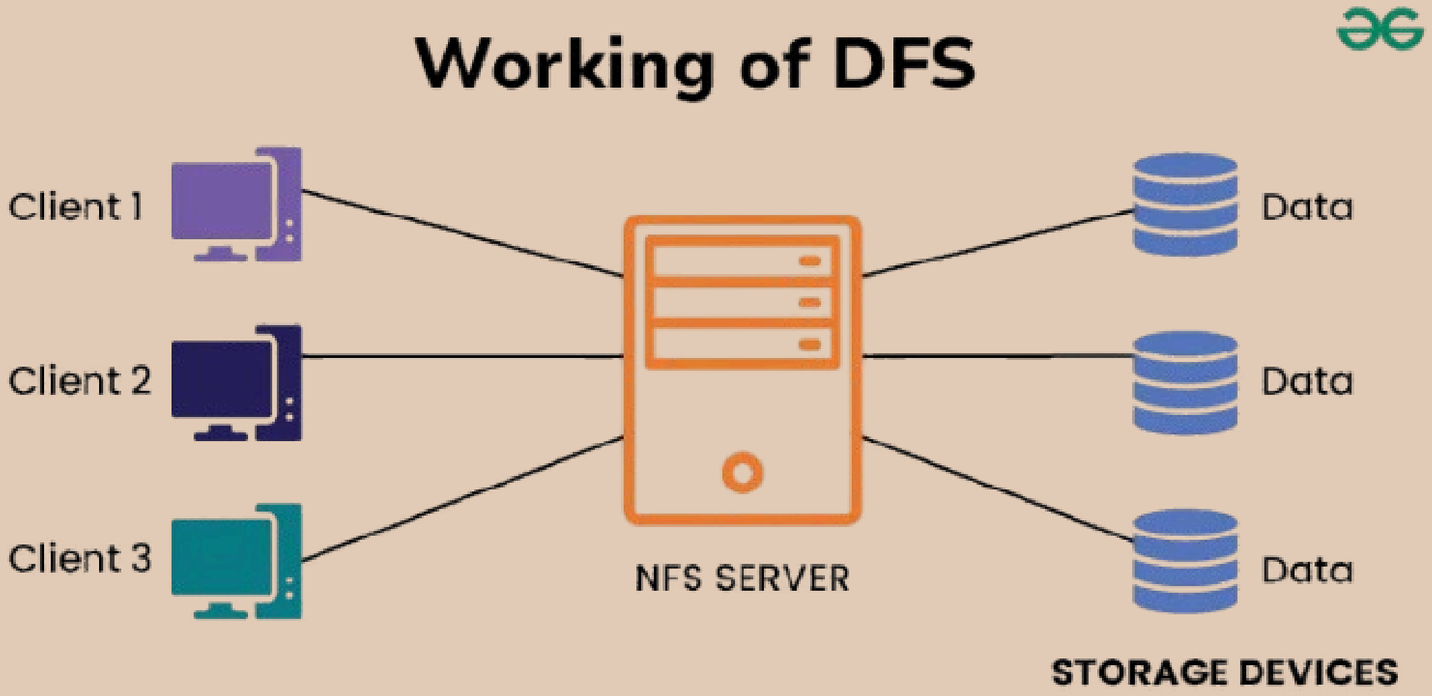
PERFORMANCE SCOPE

- High Scalability
- High Availability
- Proper Load Balancing

Exactly one semantics for Record Append in GFS

INTRODUCTION TO DFS

A Distributed File System (DFS) is a storage system that manages files across multiple servers while presenting them as a unified file system to users and applications.



GOOGLE FILE SYSTEM

The Google File System (GFS) is a scalable, distributed file system designed by Google to handle large-scale data processing. It provides fault tolerance, high performance, and scalability to support Google's vast storage and computational needs. Following are the components of GFS:-

01 MASTER SERVER

Acts as the central coordinator of the file system, maintaining metadata and managing the namespace.

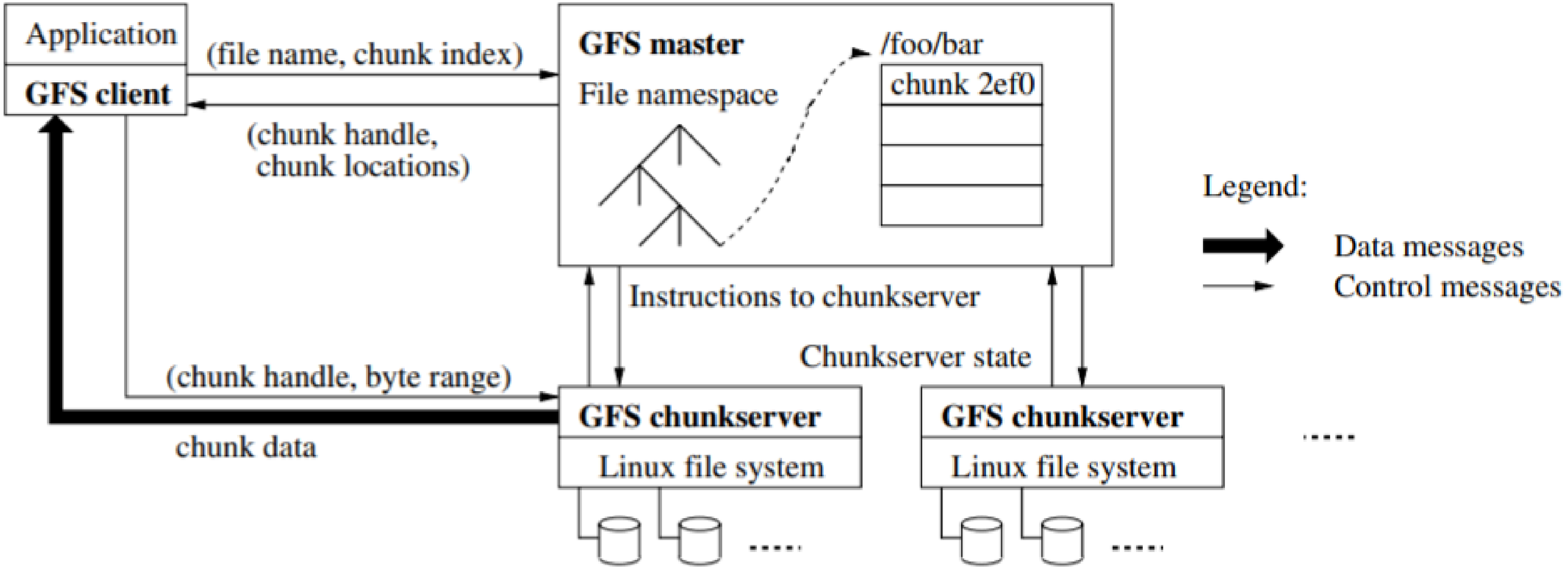
02 CHUNKSERVERS

Store data in fixed-sized chunks (64 MB by default) and handle read/write operations.

03 CLIENTS

Serve as the interface for applications to interact with the file system.

CORE ARCHITECTURE



ATLEAST ONCE VS EXACTLY ONCE SEMANTICS

ATLEAST ONE SEMANTICS

Each operation is retried if the server fails to acknowledge it.

It guarantees that the operation will be performed at least once.

Duplicate writes may occur if an operation is retried after a partial success, results into data inconsistencies.

EXACTLY ONE SEMANTICS

Each operation is performed exactly once, even in the event of failures.

Ensures that operation is done exactly once, avoidance of duplicate data writes.

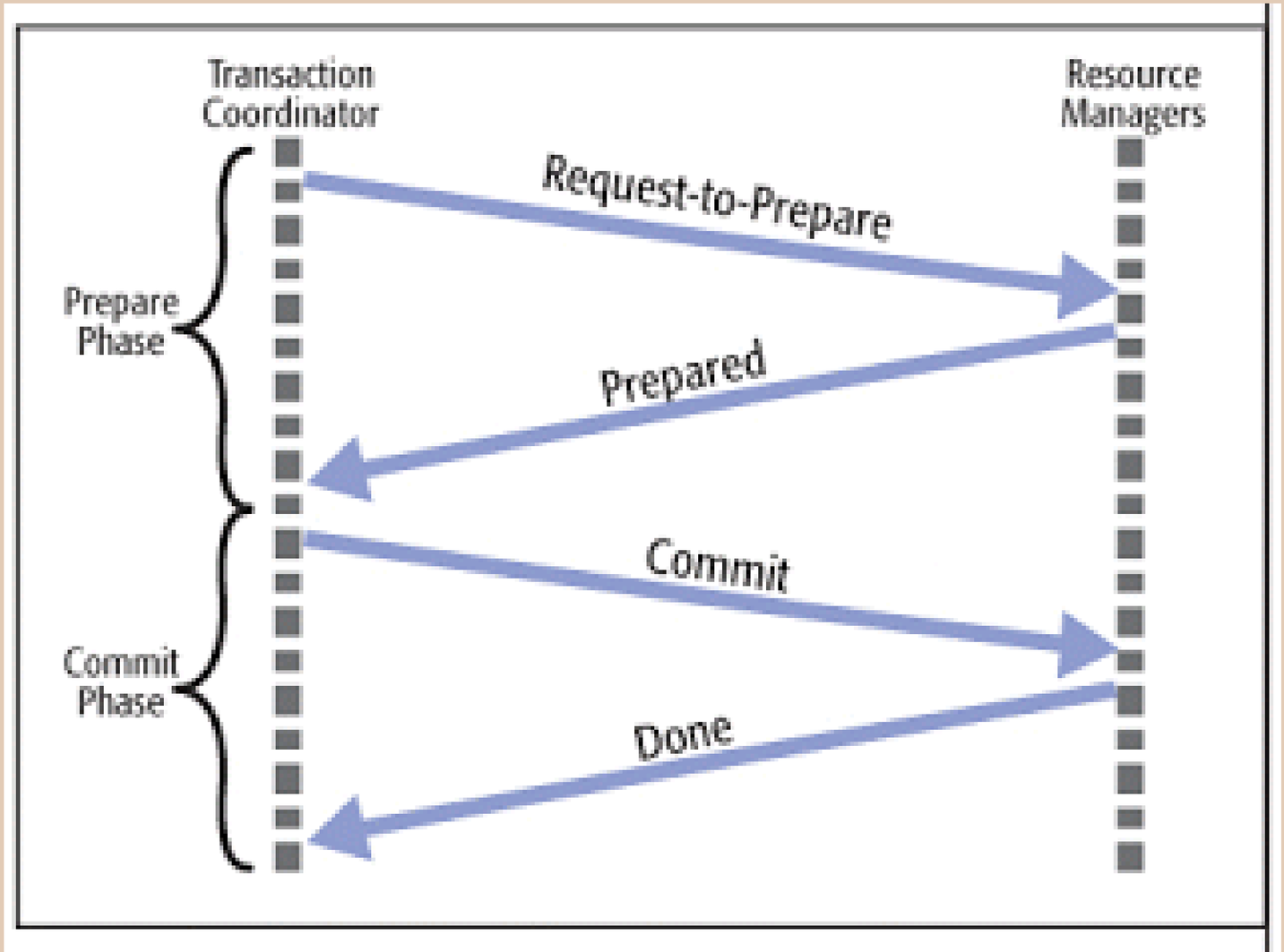
Stronger consistency guarantees across distributed nodes.

IMPLEMENTATION OF EXACTLY ONCE SEMANTICS

Unique Message Identification
Assign a unique ID (e.g., UUID or timestamp) to every operation or message.
Maintain a deduplication log at the receiver (e.g., chunk server) to track already processed IDs, ensuring idempotent operations.

Commit Protocol:
Use a two-phase commit (2PC) to ensure all replicas agree on the operation.
Phase 1: Prepare the operation (ensure all replicas can apply it).
Phase 2: Commit the operation (apply it atomically across replicas)

2 PHASE COMMIT PROTOCOL



2 PHASE COMMIT PROTOCOL

Prepare Phase

Generate a unique transaction ID.
Check if all chunk servers have sufficient storage, can replicate data, and have no conflicts.
Proceed only if all nodes confirm readiness to prevent partial updates.

Commit Phase

Append data to all chunk servers synchronously.
Ensure atomicity, where all nodes either commit or abort together.
Update the master server’s metadata and roll back on failure.

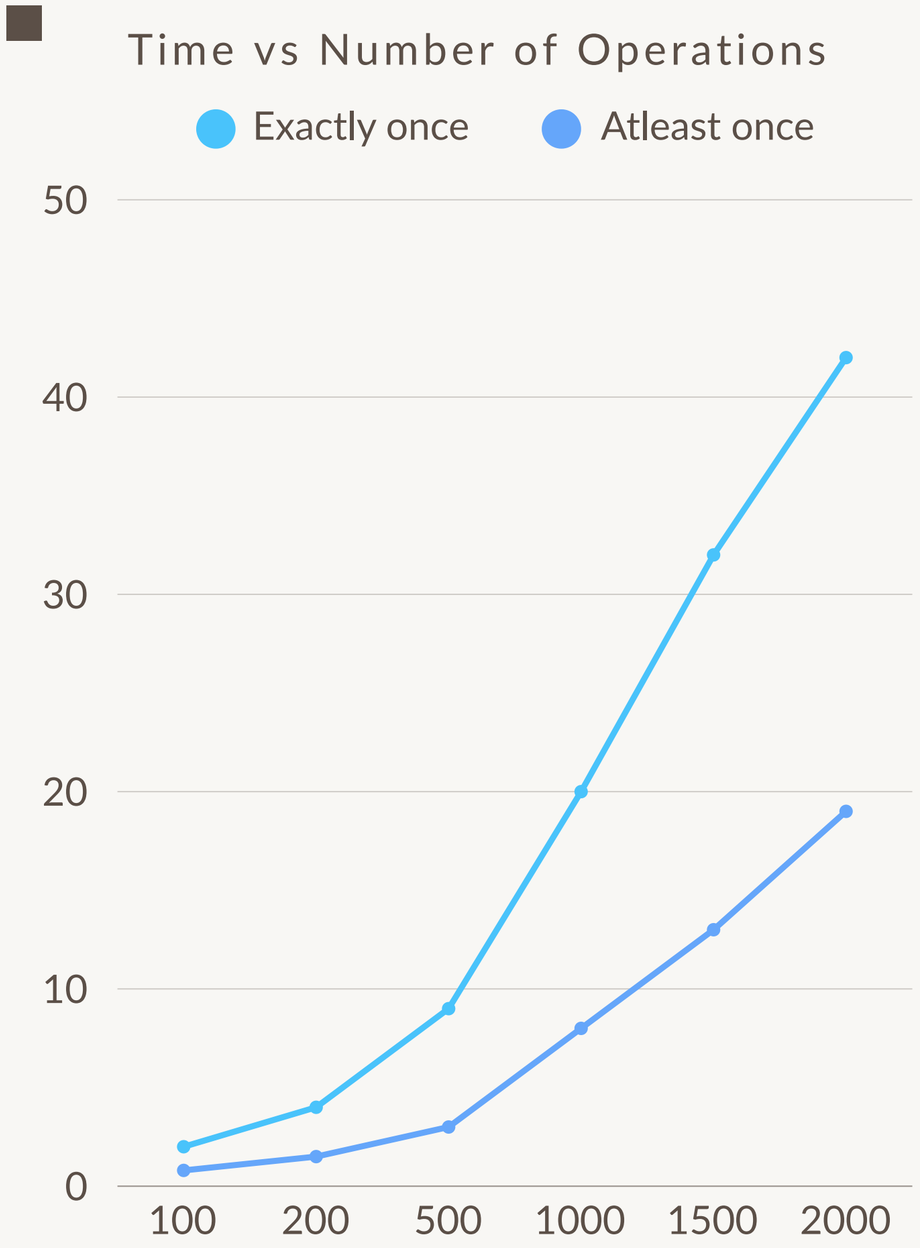
OPERATIONS IN FILE SYSTEM

- Upload Operation: The client uploads a file by dividing it into chunks and sending each chunk to a set of chunk servers. The master server coordinates the process and maintains metadata about the file and its chunks.

- Read Operation: The client sends a read request to the master server, which redirects it to the appropriate chunk servers.

- Write Operation: Writes are handled similarly to uploads but may involve updating existing chunks. The primary replica manages the write operation, ensuring consistency across replicas through a synchronized commit process.

Record Append: The record append operation allows clients to append data to a file. This is especially useful for log files and other append-only datasets



RESULTS

Based on current implementation of both semantics:

Exactly Once

15–30 ms per operation

Atleast Once

5–10 ms per operation

Difference caused due to

- Coordination between master and chunkservers.
- Execution of the 2PC protocol to ensure atomicity.
- Synchronization across replicas to ensure consistency.

LIMITATIONS

Overhead of 2PC commit protocol

For reads, randomly one of the servers in chosen(random load balancing), instead can use dynamic methods to detect best server

Using 2PC for write

Further Improvements

Better Load Balancing

Prevent hotspots by distributing file chunks more evenly

Dynamic Scaling

Automate chunkserver addition and data rebalancing

Support for Write Operations

Currently supports exactly one semantics append-only operations, which can be extended to writes too.

Better Fault Tolerance

Implement finer fault tolerance mechanisms at the chunk or file level

OBSERVATION

AT LEAST ONCE SEMANTICS

Faster but risks data duplication, which can complicate application logic.

EXACTLY ONCE SEMANTICS

Slower due to overhead but ensures data consistency and simplifies application behavior.

CONCLUSION

Exactly Once semantics addresses the challenges of data duplication and application failure by leveraging the response time.

The project provides a scalable, resilient, and high-performance solution inspired by the success of GFS.

THANK YOU