THE UNIVERSITY *of* EDINBURGH

Distributed Systems Fall 2024

Yuvraj Patel

Today's Agenda

Google File System (GFS)

Coursework is released

Google File System (GFS)

Scalable, distributed file system for large distributed data-intensive applications at Google

Based on a different set of assumptions leading to different design choices than conventional file systems

Implemented as user-space library

Goals

- Large storage
- Scalable access
- Fault tolerance
- Transparency (location and fault)

GFS Assumptions

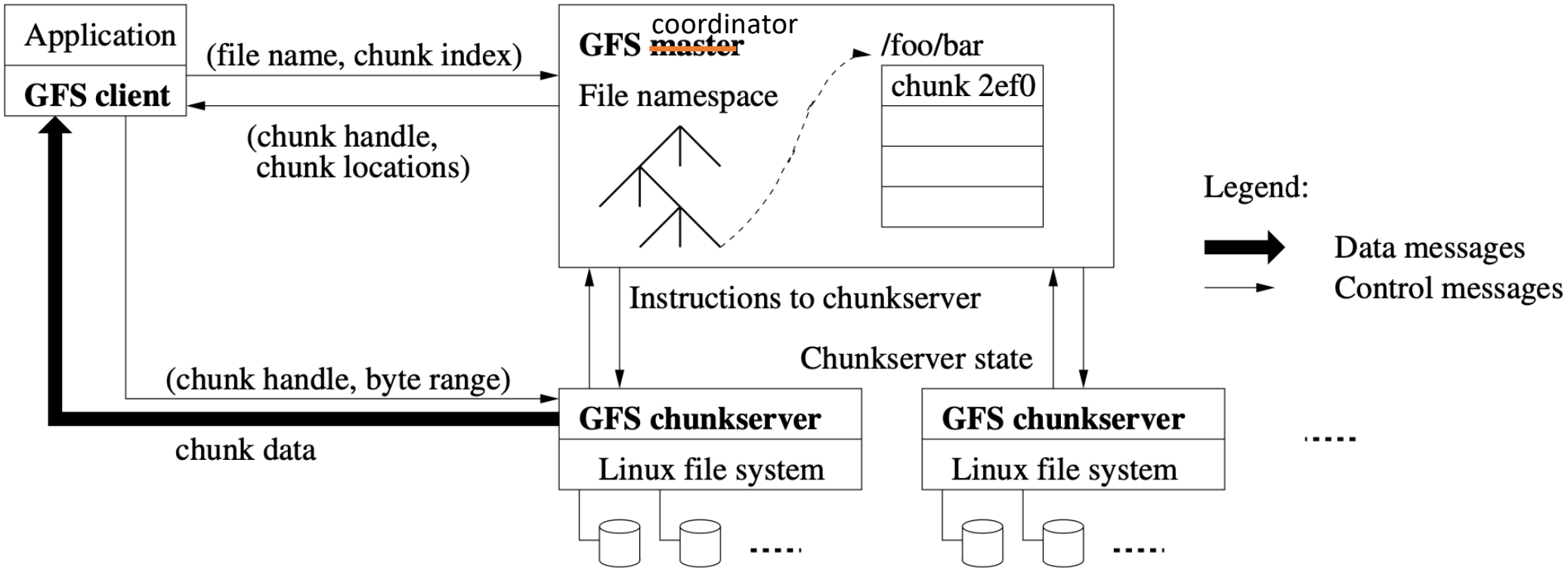
Hardware failures are common

Files are large (GB/TB); millions of files

Files access

- Most files are appended, not overwritten; Large appends
- Random writes within a file are almost never done
- Files are mostly read; often sequentially
- Two types of reads – Large streaming reads and small random reads
- Concurrent appends possible

GFS Architecture



GFS Architecture – Coordinator

Single Coordinator

- State replicated on backups

Holds all metadata

- Namespace
- Access-control
- Filename to Chunk Mapping
- Chunk locations

All metadata stored in the main memory

GFS Architecture – Coordinator (contd…)

Manages

- Chunk leases to chunkservers
- Garbage Collection of orphaned chunks
- Chunk migration (copying/moving chunks)

Fault Tolerance

- Periodically communicates with all chunkservers via heartbeats
- Operation log replicated on multiple machines

GFS Architecture – Chunk & Chunkservers

Chunk size = 64 MB

Chunkserver

- Stores chunks on local disks as normal files
- Stores a 32-bit checksum with each chunk to detect corruption

Each chunk replicated (3 replicas) on multiple chunkservers

Chunk Handle to identify a chunk

- Globally unique 64-bit number
- Assigned by the coordinator when the chunk is created
- Read/write requests specify chunk handle and byte range

GFS Files Viewpoint

GFS Clients

Hundreds/Thousands of clients

Issues

- Control requests to coordinator
- Data requests directly to chunkservers

Caches metadata

No caching of data

No OS-level API; instead use library (GFS client code linked into each application)

GFS Client Read

GFS Client Write

One chunkserver is primary for each chunk

Coordinator grants lease to primary (60 sec)

Leases renewed using periodic heartbeat messages between coordinator and chunkservers

Primary chooses the order for all client writes

- Tells the secondaries – with sequence numbers – so all replicas apply writes in the same order, even for concurrent client writes

GFS Client Atomic Record Append

GFS provides an atomic append operation called record append

Unlike traditional writes, client specifies only the data

GFS appends it to the file at least once atomically at an offset of GFS's choosing and returns that offset to the client

- Like writing to a file opened in O_APPEND mode in Unix without race conditions when multiple writers do so concurrently

GFS Client Atomic Record Append (contd…)

Same control flow as writes

Client pushes data to replicas of last chunk of file

Client sends request to primary

Request fits in current last chunk

- Primary appends data to own replica
- Primary tells secondaries to do same at same byte offset in theirs
- Primary replies with success to clients

GFS Client Atomic Record Append (contd…)

If data does not fit in the last chunk

- Primary fills current chunk with padding
- Primary instructs secondaries to do the same
- Primary replies client to retry the operation on next chunk

If record append fails at any replica, client retries operation

- Replicas of same chunk may contain different data including duplicates of all or part of record data

GFS Metadata Consistency & Operation Log

Changes to namespace are atomic

Coordinator uses operation log to store critical metadata changes

Log defines a timeline that defines the order of concurrent operations

Log stored on coordinator's local disk and replicated on remote machines

Coordinator recovers its file system state by replaying the operation log

Coordinator only replies to client after log entries safe on local disk and replicas

GFS Consistency Model – Data

Defined – If primary tells client that a write succeeded, and no other client is writing the same part of the file, all readers will see the write

Consistent – If successful concurrent writes to the same part of a file happens, and they all succeed, all readers will see the same content, but maybe it will be a mix of the writes

Inconsistent – If primary doesn't tell the client that the write succeed, different readers may see different content, or none

	Write	Record Append
Serial success	*defined*	*defined* interspersed with *inconsistent*
Concurrent successes	*consistent* but *undefined*	
Failure	*inconsistent*	

GFS Consistency Model – Data (contd…)

How can inconsistent content arise?

- Primary updates its own state but one secondary did not update (slow, fail)
- Client1 reads from P; Client 2 reads from S1
- Both clients will see different results

How can consistent but undefined arise?

- Clients break big writes into multiple small writes (at chunk boundaries), and GFS may interleave them if concurrent client writes happen

How can duplicate data arise?

- Clients re-try record appends

Applications must cope with the data inconsistency

GFS Applications & Record Append Semantics

Applications rely on checksum in records they write using Record Append

Readers can identify padding and record fragments using checksum

If application cannot tolerate duplicates, they can use unique identifiers in the records

Readers can use unique identifies to identify and filter duplicates

GFS Stale Replica Detection

For each chunk, coordinator maintains a chunk version number to distinguish between up-to-date and state replicas

Coordinator increase chunk version number and informs the up-to-date replicas when the coordinator grants a new lease on a chunk

When chunkserver restarts/recovers after crash, on reporting its set of chunks and their version numbers, coordinator can identify stale replicas

Coordinator removes state replicas in its regular garbage collection

Coordinator also passes the version number to client; clients can check the version numbers to ensure it is accessing up-to-date data

GFS – Handling faults

Secondary

- Primary may retry “n” times
- Client can retry
- Coordinator may remove from chunkhandle lists, replicate chunk data

Primary

- Coordinator may remove from chunkhandle lists
- Coordinator will grant lease to any secondary

Coordinator

- Replay operation log, rebuild state, resume operations
- Ask chunkservers what they store
- Wait for one lease time before granting lease to any secondary