THE GOOGLE FILE SYSTEM*

CSC 256/456 - UNIVERSITY OF ROCHESTER JING ZHANG

*GHEMAWAT, SANJAY, HOWARD GOBIOFF, AND SHUN-TAK LEUNG. "THE GOOGLE FILE SYSTEM." ACM SIGOPS OPERATING SYSTEMS REVIEW. VOL. 37. NO. 5. ACM, 2003.

Motivation

- Common design goals
  - Performance, scalability, reliability & availability
- Design driven by Google's application workloads and technological environment
  - Component failures are the norm rather than the exception.
    - Fault tolerance, automatic recovery …
  - Files are huge by traditional standards.
    - Block size assumption …
  - Record appends are prevalent than random writes.
    - Focus on "appending" for performance optimization

Design Overview --Architecture

- One Master, multiple Chunkservers, accessed by multiple clients.
- Files are divided into fixed size chunk. (64MB)
- Global unique "chunk handle" (64bit)
- Replication on multiple chunkservers

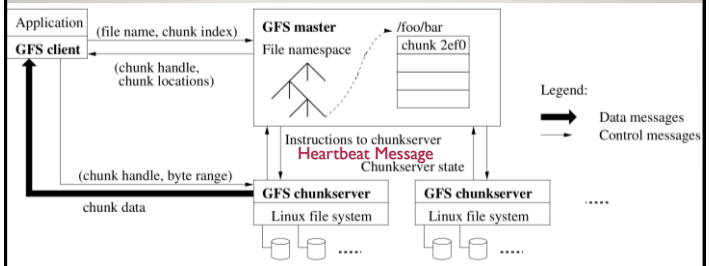


Figure 1: GFS Architecture

Design Overview --Architecture

- Master:
  - Maintain all file system metadata.
    - Namespace, access control information, mapping from files to chunks, chunk locations ….
  - Send ***Heartbeat Messages*** to chunkservers periodically
    - Instructions, state monitoring.
- Client:
  - Communicate with Master for metadata.
    - Clients usually cache master's replies (chunk handle & locations of replicas)
  - Communicate with Chunkservers for data read/write.

Design Overview
--Architecture

Comparison with AFS/NFS

AFS/NFS	GFS
- Single level architecture - Sever also contains data	- Two levels architecture - Separate control and data flow

Single Master is a simpler design, but need to avoid bottleneck:
- Minimize Master's involvement in reads & write.
  → Separate control flow & data flow.
- Speed up Master's operations.
  → Metadata is all in-memory.

Design Overview
-- Metadata

- Metadata includes:
  - File and chunk namespaces
  - Mapping from files to chunks
  - Location of each chunk replicas
- All metadata is stored in Master's memory
- Master does **NOT** keep a persistent record of chunk locations. *Why?*
- Operation Log
  - Timeline for the order of concurrent operations
  - Stored on Master's **Local Disk** and replicated on remote machines
  - Recover file system state by replaying operation logs *Take too long to recover?*
    - Checkpoints

Design Overview
-- Interface

- Common file operations:
  - Create, delete, read, write, open & close.
- Moreover:
  - Snapshot (copy-on-write)
  - Atomic Record append

System Interactions
Lease & Mutation Order

- Each mutation should be performed at all replicas.
- Master grants a ***Lease*** to one of the replicas, making it the ***primary*** replica.
- Primary picks a serial order for replicas to be mutated.
- Lease has an initial timeout of 60 seconds.
  - Can be extended. *How? Heartbeat Message*
  - Or can be revoked before expiration.

System Interactions
-- Dataflow

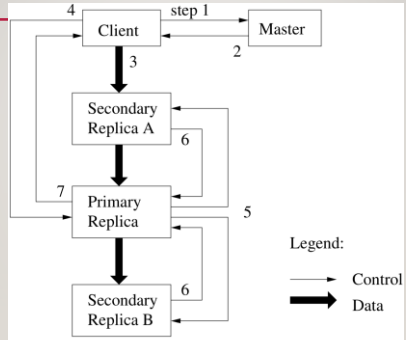


Figure 2: Write Control and Data Flow

System Interactions
-- Atomic ***Record Appends***

- Client specify only the data, GFS choose the offset when appending data to files.
- Primary replica check if the data size would cause chunk-size overflow.
  - If so, it pads the chunk to the max size, tells secondary to do the same; and replies to Client to retry on next chunk.
- If coming across failures, retry. → Duplication in some replicas.

Relaxed Consistency Model

- A file region is ***consistent*** if all clients will always see the same data, regardless of which replicas they read from.
- A region is ***defined*** after a file data mutation if it is consistent and clients will see what the mutation writes in its entirety.
- Write vs. Record Append

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

Table 1: File Region State After Mutation

Try write again when coming across failure.

↓

Duplication

Namespace Management
-- Locking

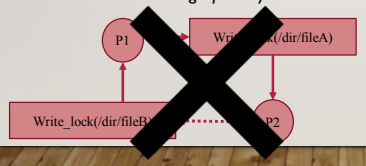
- Lookup table mapping full pathname to metadata.
- Each absolute file name or directory name has an associated read-write lock.
- Example: create /dir1/dir2/file

  read_lock (/dir1)
  read_lock (/dir1/dir2)
  write_lock (/dir1/dir2/file)
  create file
  unlock (/dir1/dir2/file)
  unlock (/dir1/dir2)
  unlock (/dir1)

- Allow concurrent mutations in the same directory.
- By locking new file before it is created: no file with same name will be created simultaneously.

Namespace Management -- Deadlock Prevention

- Deadlock Prevention *Which one to break for GFS?*
  - Recall: 4 necessary conditions
- Google file system
  - First ordered by *level* in the namespace tree
  - *Lexicographically* within the same level



Circular Wait Condition Not Allowed

Fault Tolerance -- High Availability

- Fast Recovery
  - Log & Checkpoint
- Replication
  - Chunk replication
    - Each chunk has 3 replicas by default.
    - Re-replication is triggered once the number of replicas fall below a user-specified goal.
  - Master replication
    - Metadata replicas on multiple remote machine.
    - Shadow masters provides read-only access.

Summary

"The design is driven by Google's application workloads and technological environment."

- A file system designed only for Google
  - Huge files → Huge chunk size
  - High demands of record appending → Atomic ***Record Append***
  - … …
- A file system might not be adoptable in other systems, but very inspiring.

Reference

[1] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS Operating Systems Review. Vol. 37. No. 5. ACM, 2003.

[2] https://www.slideshare.net/romain_jacotin/the-google-file-system-gfs