Google File System Notes

The Google File System (GFS) is a scalable, distributed file system.

Overview

- GFS is designed for files that are 100 MB or larger. Multi-GB files are common. GFS is not optimized for small files.
- Writes are typically large, sequential writes that are appended to files. Once written, files are seldom modified again.
- Most client applications prioritize processing data in bulk at a high rate. Not many have low latency requirements for an individual read/write.

Design

Each GFS cluster has a single master and multiple chunk servers (3 replicas at default).

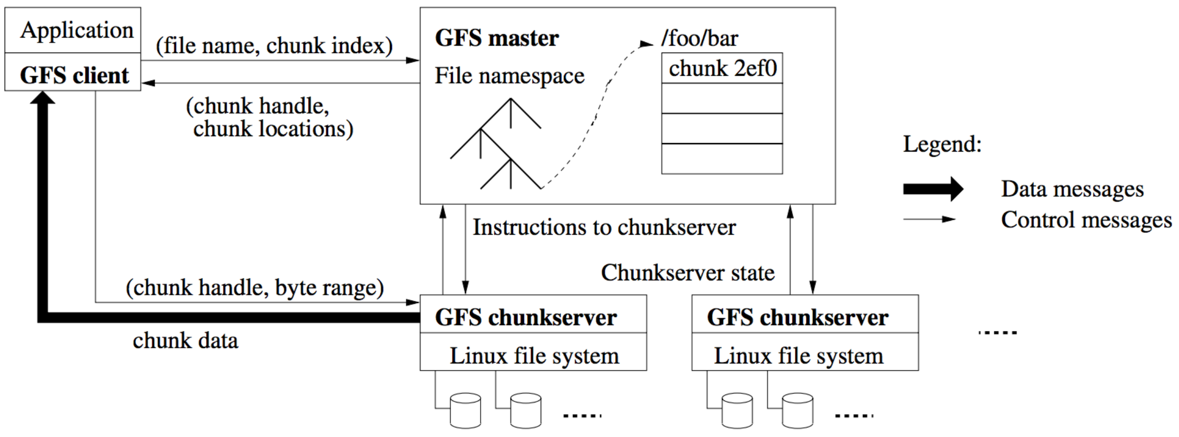

Figure 1: GFS Architecture

Master

- In terms of data, the master is solely responsible for the file system metadata. The master is not involved in data mutation operations--the most common operation. This allows for a simple, centralized master that does not become a bottleneck.
- The master sends HeartBeat messages to each chunkserver to provide instructions and collect state.
- As metadata is minimal compared to chunks, the master is able to keep all metadata in memory. This allows master operations to be fast.

Chunkservers

- Chunkservers do not cache file data. Files are too large to be cached.
- Primary chunkservers are responsible for data mutations and replicating to secondary replicas.

Clients

- Clients initially interact with the master for metadata. After that, clients communicate directly with the chunkservers
- Clients only cache metadata. Clients stream huge files or have working sets too large to be cached.

Chunk Size

- Files are divided into fixed-size 64 MB chunks. Each chunk is identified by a globally-unique 64 bit chunk handle.
- Large chunk sizes reduce client interactions with the master. Reads and writes on the same chunk only require one initial request to the master for chunk location info.

Metadata

- The metadata consists of file and chunk namespaces, access control information, file-to-chunk mappings, and chunk locations (on all replicas).
- All metadata is available in master's memory.
- Namespaces and file-to-chunk mappings are persisted through log mutations on an operation log stored on master's local disk. The operation log is also replicated on remote machines. During master startup, it replays the operation log to recover its file system state.

Chunk Locations

- Chunk locations are not persistently stored on master's disk. Chunkservers provide chunk location information to the master. When a master starts up, it polls the chunkservers for this info. When a chunkserver joins a cluster, it provides this info to the master.
- The master stays up-to-date on chunk locations through periodic HeartBeat messages.

Operation Log

- When the operation log becomes large, the master will checkpoint its state.
- Master startup time is minimized by keeping the log small and replaying from the last checkpoint.
- The checkpoint is a compact B-tree. Older checkpoints can be deleted after the creation of a new checkpoint. A few checkpoints are kept to safeguard against catastrophes.

Leases and Mutation Order

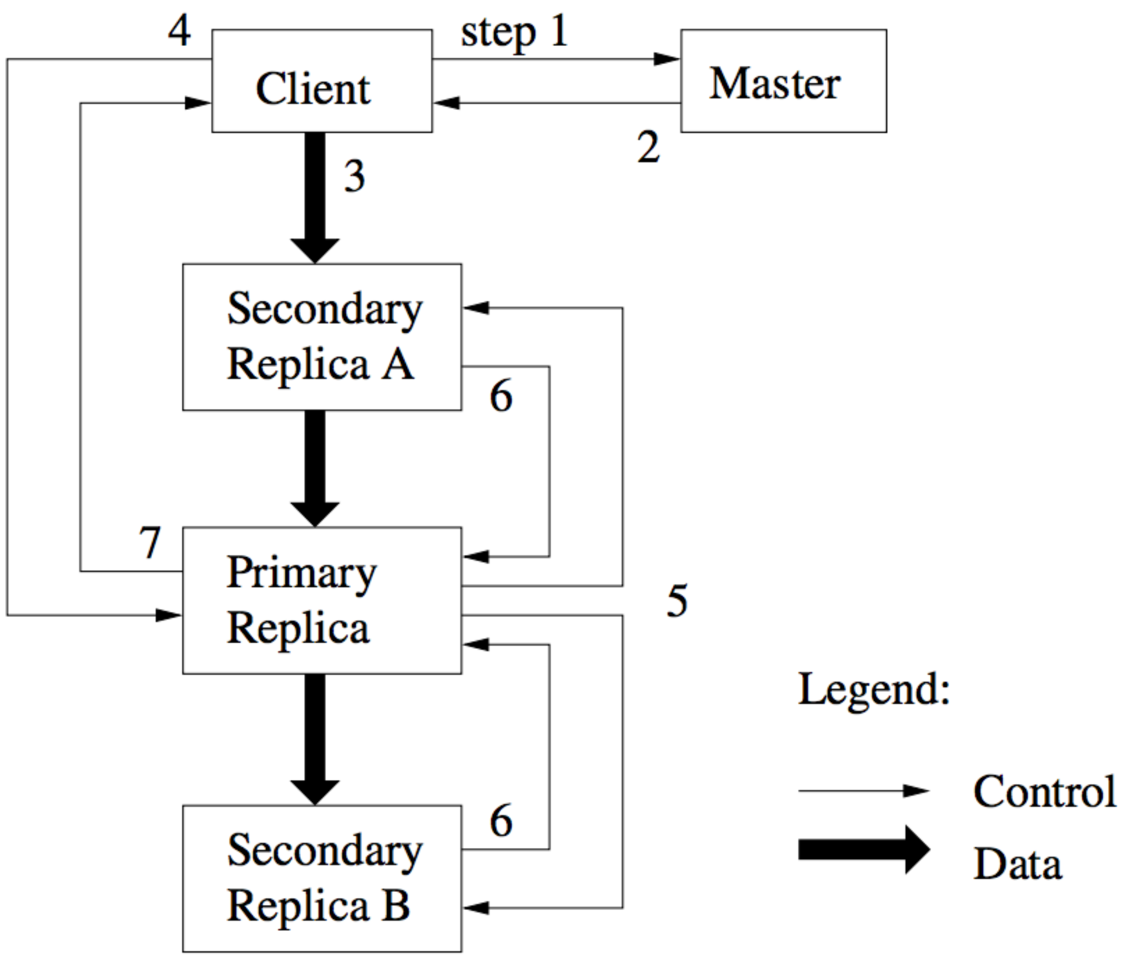

Figure 2: Write Control and Data Flow

These are the steps in which mutations are applied.

1. The client requests for the primary chunkserver that hold a lease to a chunk as well as the replicas. If a chunkserver doesn't have the lease, the master grants a lease to one of the replicas. This replica becomes the primary.

2. The master replies with the identity or the primary and the locations of the secondary replicas. The client caches this lease for future mutations until the lease times out.
3. **Data Flow.** The client pushes the data to all the replicas. Each chunkserver stores the data in an internal LRU buffer cache. The send order does not matter.
4. **Control Flow.** Once all the replicas acknowledge receiving the data, the client sends a write request to the primary. The primary assigns a consecutive serial number to the mutation (serialization). It applies the mutation in its own local state in serial number order.
5. The primary forwards the write request and serial number to all the secondary replicas. Each replica applies the mutation in the same serial number order as the primary.
6. The secondary replicas reply to the primary indicating the write operation was completed (along with any errors).
7. The primary replies to the client. Any errors encountered are reported. If the write failed, the client will retry the operation from step 1.

Latency is minimized by pipelining data transfers over TCP connections. Once a chunkserver receives some data, it immediately begins forwarding to replicas. Sending data immediately does not reduce the receive rate due to using a switched network with full-duplex links. 1 MB is typically distributed in about 80 ms.

Consistency Model

- GFS has a relaxed consistency model.
- Mutations to a chunk are applied in the same order across all replicas.
- Chunk version numbers are used to detect replicas that have become stale. Stale replicas are garbage collected at the earliest opportunity.
- Whenever a master grants a new lease on a chunk, it increases the chunk version number and informs the replicas. Chunk version numbers are applied on master and replicas prior to notifying the client of the new lease.
- Leases include the chunk version number. Clients and chunkservers verify the version number prior to performing an operation.
- The master detects data corruption on chunkservers through checksumming.
- GFS accommodates the relaxed consistency model by relying on appends versus overwrites, checkpointing, and writing self-validating, self-identifying records.
- Each record prepared by the writer contains checksums so the record's validity can be verified.

High Availability

- The master and chunkservers are designed to restore their state and start up in a matter of seconds.
- The master clones existing replicas as needed. This is to keep chunks replicated as chunkservers go offline or are corrupted.
- The master's operation log and checkpoints are replicated on multiple machines.
- Master state mutations are only reported as successful once they have been recorded to disk locally and to all master replicas.

Data Integrity

- Each chunkserver verifies the integrity of its data by maintaining checksums.
- Each chunk is broken up into 64 KB blocks. Each block has a 32 bit checksum. Like other metadata, checksums are kept in memory and stored persistently.
- On each read, chunkservers verify data blocks with the checksums that overlap with the read range. This is done prior to returning data to the requestor (client or another chunkserver).
- If a block doesn't match the checksum, chunkserver returns an error to the requestor and reports a mismatch to the master.
- On error, the client will request data from another replica. The master will clone the chunk from another replica and instruct the chunkserver to delete the corrupted chunk.
- Checksumming has little effect on read performance. Checksums are only a small amount of extra data for verification.
- During idle periods, chunkservers verify data blocks of inactive chunks. This solves for detecting corruption in chunks that are rarely used.