The Google File System

Neelam Singh

Design consideration

- Built from cheap commodity hardware
- Expect large files: 100MB to many GB
- Support **large streaming reads and small random reads**
- Support **large, sequential file appends**
- Support producer-consumer queues for many-way merging and file atomicity
- Sustain high bandwidth by writing data in bulk

Interface

- Interface resembles standard file system – hierarchical directories and pathnames
- Usual operations
  - create, delete, open, close, read, and write
- Moreover
  - Snapshot – Copy
  - Record append – Multiple clients to append data to the same file concurrently

Architecture

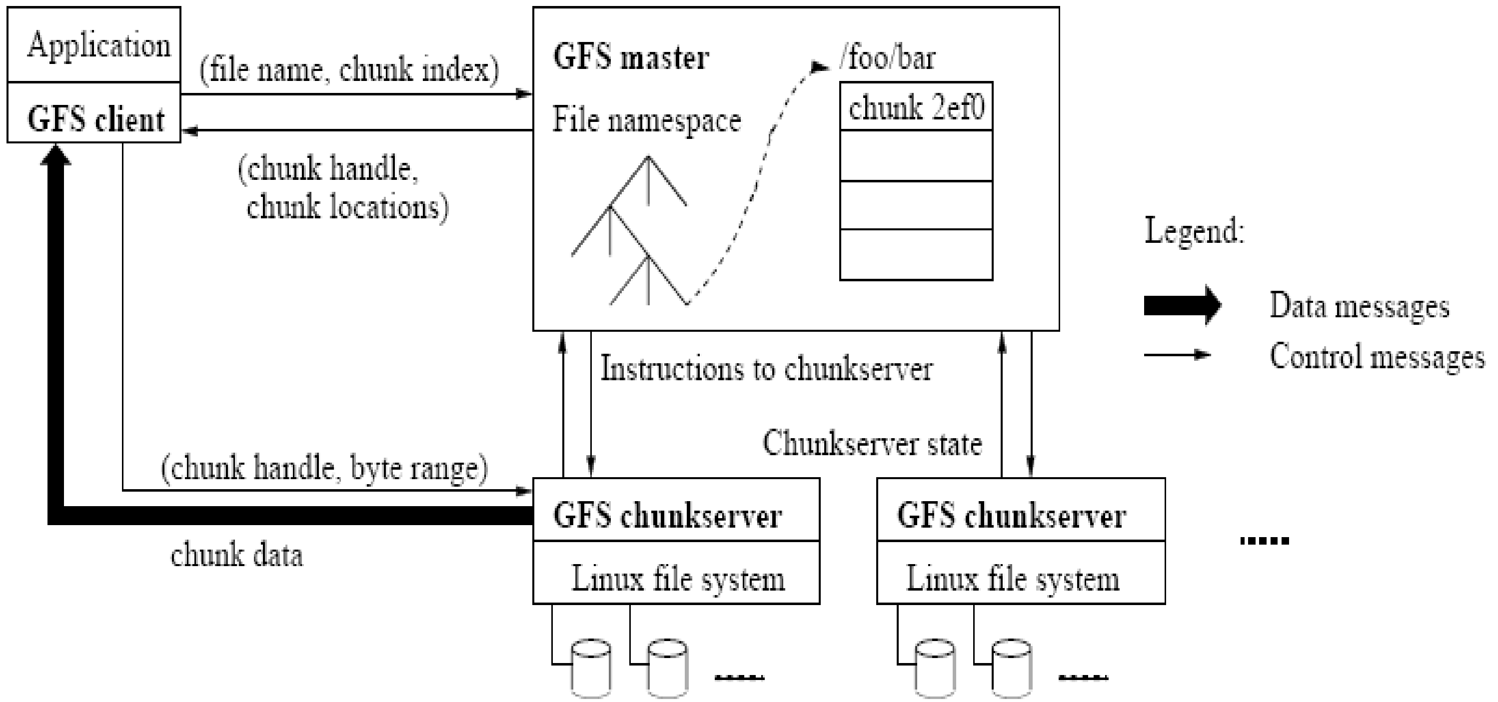

Figure 1: GFS Architecture

Chunk Size

- 64MB
  - Much larger than typical file system block sizes
- Advantages from large chunk size
  - Reduce interaction between client and master
  - Client can perform many operations on a given chunk
    - Reduces network overhead by keeping persistent TCP connection
  - Reduce size of metadata stored on the master
    - The metadata can reside in memory

Metadata (1)

- Store three major types
  - Namespaces
    - File and chunk identifier
  - Mapping from files to chunks
  - Location of each chunk replicas
- **In-memory** data structures
  - Metadata is stored in memory
  - Periodic scanning entire state is easy and efficient

Metadata (2)

- Chunk locations
  - Master do not keep a persistent record of chunk locations
  - Instead, it simply polls chunkservers at startup and periodically thereafter (heartbeat message)
  - Because of chunkserver failures, it is hard to keep persistent record of chunk locations
- Operation log
  - Master maintains historical record of critical metadata changes
  - Namespace and mapping
  - For reliability and consistency, replicate operation log on multiple remote machines

Client operations
Write(1)

- Some chunkserver is primary for each chunk
  - Master grants lease to primary (typically for 60 sec.)
  - Leases renewed using periodic heartbeat messages between master and chunkservers
- Client asks master for primary and secondary replicas for each chunk
- Client sends data to replicas in daisy chain
  - Pipelined: each replica forwards as it receives
  - Takes advantage of full-duplex Ethernet links

Client operations Write(2)

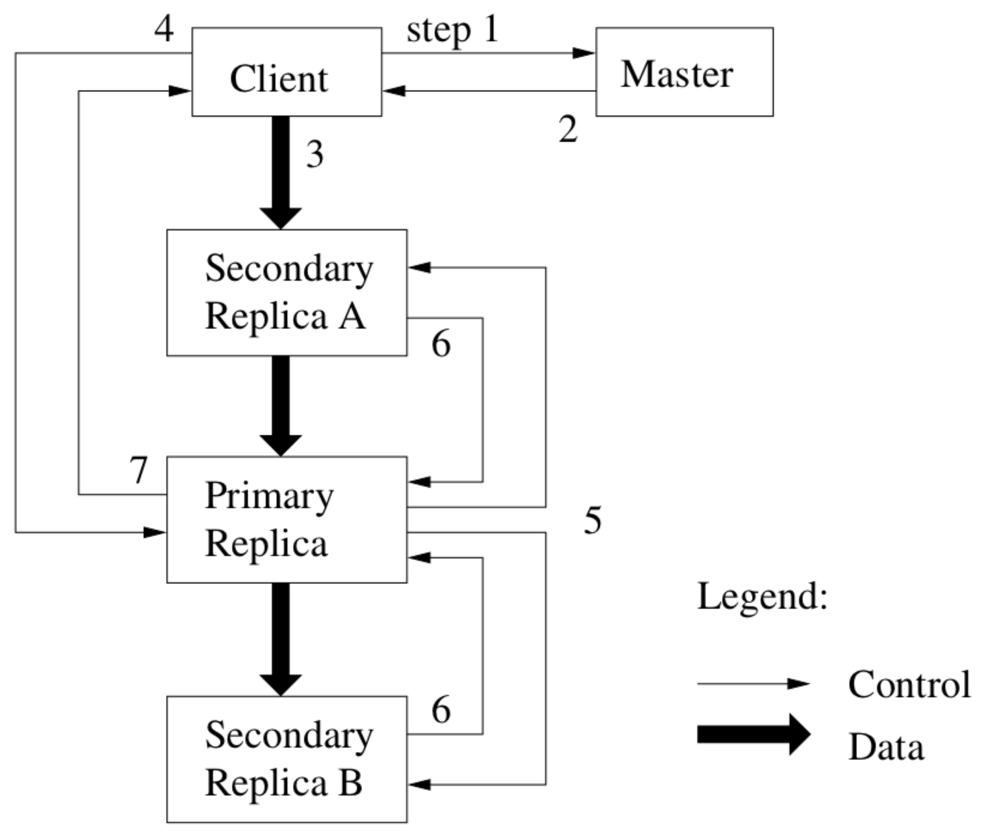

Figure 2: Write Control and Data Flow

Client operations
Write(3)

- All replicas acknowledge data write to client
- Client sends write request to primary
- Primary assigns serial number to write request, providing ordering
- Primary forwards write request with same serial number to secondaries
- Secondaries all reply to primary after completing write
- Primary replies to client

Client operations with Server

- Issues control (metadata) requests to master server
- Issues data requests directly to chunkservers
- Caches metadata
- Does no caching of data
  - No consistency difficulties among clients
  - Streaming reads (read once) and append writes (write once) don’t benefit much from caching at client

Decoupling

- Data & flow control are separated
  - Use network efficiently- pipeline data linearly along chain of chunkservers
  - Goals
    - Fully utilize each machine’s network bandwidth
    - Avoid network bottlenecks & high-latency
  - Pipeline is “carefully chosen”
    - Assumes distances determined by IP addresses

Atomic Record Appends

- GFS appends data to a file at least once atomically (as a single continuous sequence of bytes)
- Extra logic to implement this behavior
  - Primary checks to see if appending data exceeds current chunk boundary
  - If so, append new chunk padded out to chunk size and tell client to try again next chunk
  - When data fits in chunk, write it and tell replicas to do so at exact same offset
  - Primary keeps replicas in sync with itself

Master operations
Logging

- Master has all metadata information
  - Lose it, and you've lost the filesystem!
- Master logs all client requests to disk sequentially
- Replicates log entries to remote backup servers
- Only replies to client after log entries safe on disk on self and backups!

Master operations
What if Master Reboots

- Replays log from disk
  - Recovers namespace (directory) information
  - Recovers file-to-chunk-ID mapping
- Asks chunkservers which chunks they hold
  - Recovers chunk-ID-to-chunkserver mapping
- If chunk server has older chunk, it’s stale
  - Chunk server down at lease renewal
- If chunk server has newer chunk, adopt its version number
  - Master may have failed while granting lease

Master operations
Where to put a chunk

- It is desirable to put chunk on chunkserver with below-average disk space utilization
- Limit the number of recent creations on each chunkserver- avoid heavy write traffic
- Spread chunks across multiple racks

Re-replication and Rebalancing

- Re-replication occurs when the number of available chunk replicas falls below a user-defined limit
  - When can this occur?
    - Chunkserver becomes unavailable
    - Corrupted data in a chunk
    - Disk error
    - Increased replication limit
- Rebalancing is done periodically by master
  - Master examines current replica distribution and moves replicas to even disk space usage
  - Gradual nature avoids swamping a chunkserver

Garbage Collection

- Lazy
  - Update master log immediately, but…
  - Do not reclaim resources for a bit (lazy part)
  - Chunk is first renamed, not removed
  - Periodic scans remove renamed chunks more than a few days old (user-defined)
- Orphans
  - Chunks that exist on chunkservers that the master has no metadata for
  - Chunkserver can freely delete these

Fault Tolerance

- Fast Recovery
  - Master and Chunkserver are designed to restore their state and restart in seconds
- Chunk Replication
  - Each chunk is replicated on multiple chunkservers on different racks
  - According to user demand, the replication factor can be modified for reliability
- Master Replication
  - Operation log
    - Historical record of critical metadata changes
  - Operation log is replicated on multiple machines

Conclusion

- GFS is a distributed file system that support large-scale data processing workloads on commodity hardware
- GFS has different points in the design space
  - Component failures as the norm
  - Optimize for huge files
- GFS provides fault tolerance
  - Replicating data
  - Fast and automatic recovery
  - Chunk replication
- GFS has the simple, centralized master that does not become a bottleneck

- GFS is a successful file system
  - An important tool that enables to continue to innovate on Google's ideas