The Google File System

Viet-Trung Tran

Abstract

- We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications.

outline

- Introduction – Design Point
- Design Overview
- System Interactions
- Master Operation
- Fault Tolerance
- Conclusion♪

Introduction - Design Point

- Traditional Issues
  - Performance
  - Scalability
  - Reliability
  - Availability
- Different Points in GFS
  - Component failures are the norm rather than the exception
  - Files are huge that multi-GB files are common
  - Most files are mutated by appending new data rather than overwriting existing data

Interface

- Organized hierarchically in directories and identified by path names
- 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 (1)

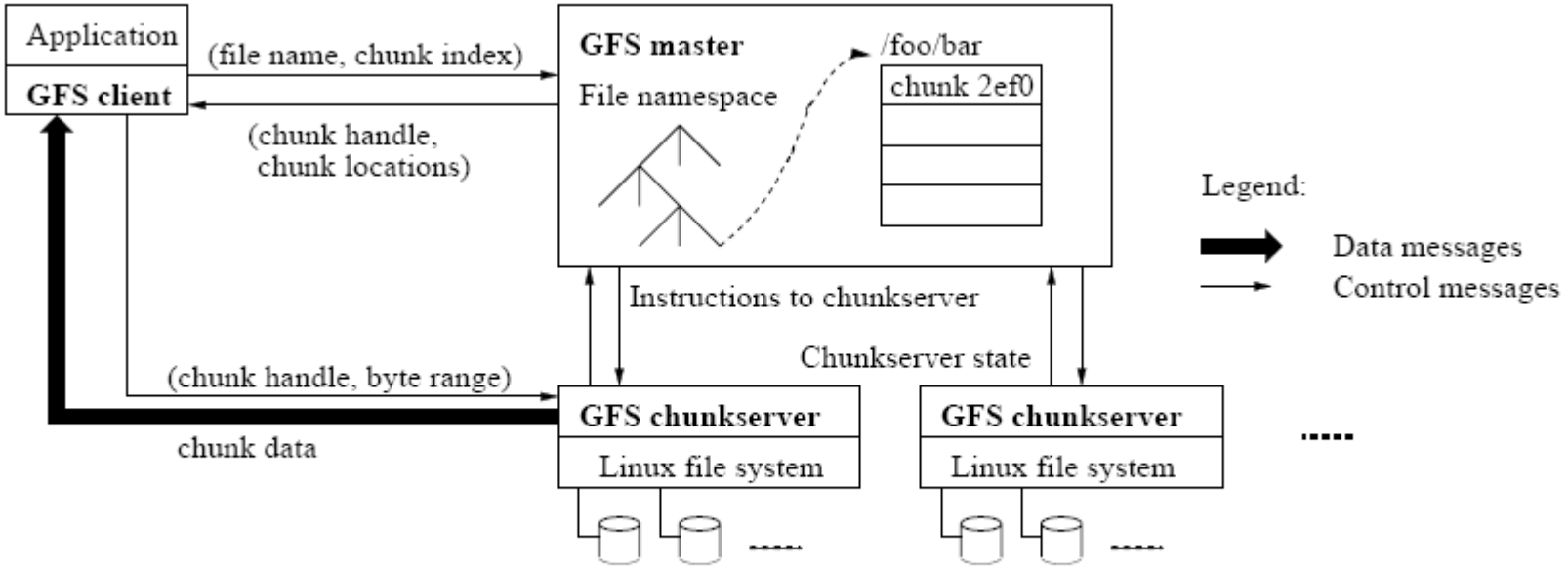

Figure 1: GFS Architecture

- Single master
- Multiple chunkservers
- Multiple clients

Architecture (2)

- Chunk
  - Files are divided into fixed-size chunks
  - Each chunk is identified by 64-bit chunk handle
  - Chunkservers store chunks on local disk as Linux files
  - Replication for reliability (default 3 replicas)
- Master
  - Maintains all file system metadata
    - Namespace, access control information, mapping, locations
  - Manage chunk lease, garbage collection, chunk migration
  - Periodically communicate with chunkservers (HeartBeat message)

Simple read procedure

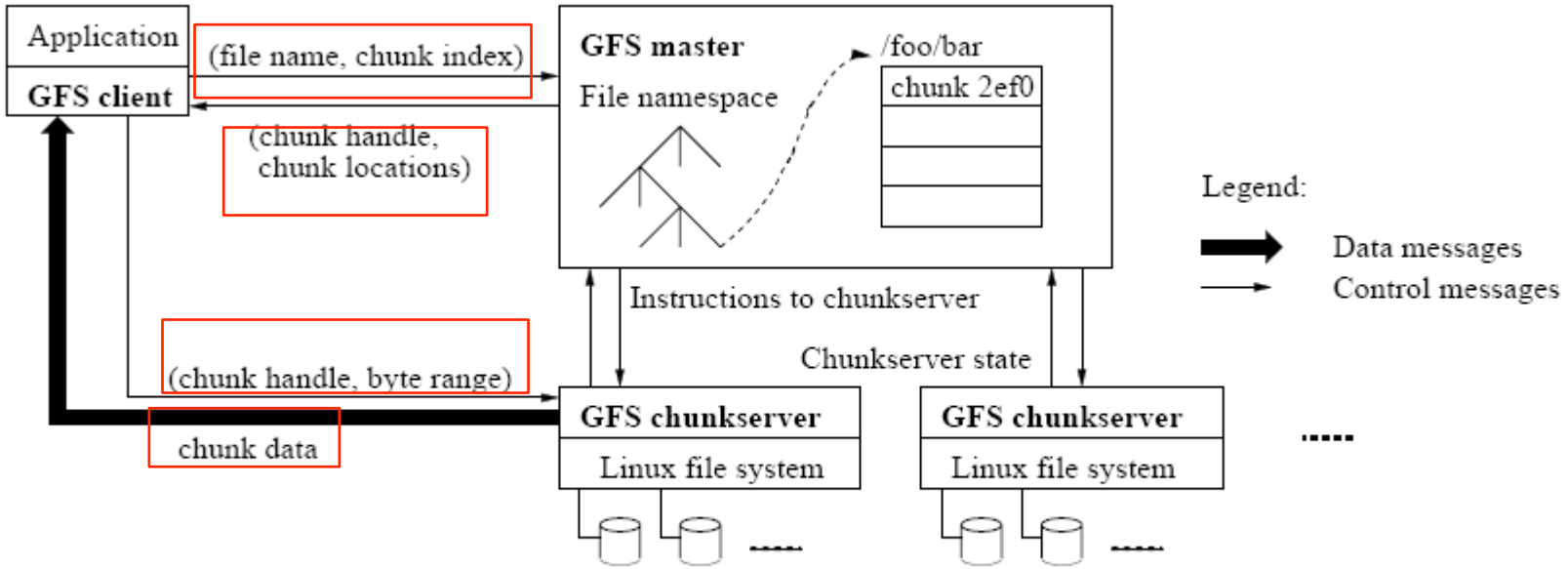

Figure 1: GFS Architecture

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

System Interactions: Leases and Mutation Order

- Use leases to maintain consistent mutation order across replicas
- Master grant lease to one of the replicas -> Primary
- Primary picks serial order for all mutations
- Other replicas follow the primary order
- Minimize management overhead at the master
- Use pipelining for fully utilize network bandwidth1

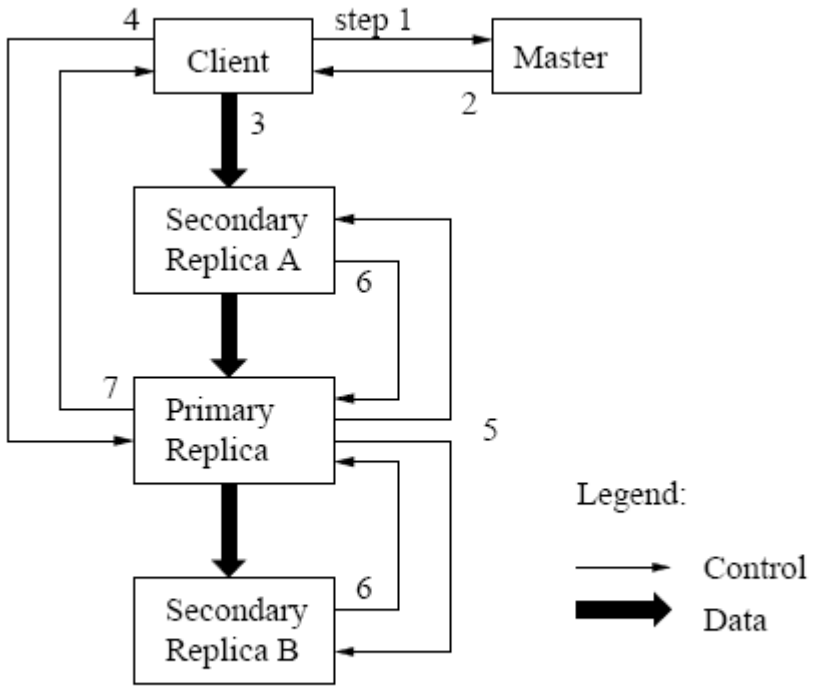

Figure 2: Write Control and Data Flow

Atomic Record Appends

- Atomic append operation called record append
- Record append is heavily used
- Clients would need complicated and expensive synchronization
- Primary checks if append exceed max chunk size
  - If so, primary pads chunk to max chunk size
  - Secondaries do the same
  - Primary replies to the client that operation should be retried on the next chunk000

Snapshot

- Make a copy of a file or a directory tree
  - Master revokes lease for that file
  - Duplicate metadata
  - On first write to a chunk after the snapshot operation
    - All chunkservers create new chunk
    - Data can be copied locally

Namespace Management and Locking

- GFS master maintain a table which map full pathname to metadata
- Each node in the namespace has associated read-write lock
- Concurrent operations can be properly serialized by this locking mechanism

Replica Placement

- GFS place replicas over different racks for reliability and availability
- Read can exploit aggregate bandwidth of multiple racks but write traffic has to flow through multiple racks
  -> need tradeoff

Creation, Re-replication, Rebalancing

- Create
  - Equalize disk space utilization
  - Limit recent creation on each chunkserver
  - Spread replicas across racks
- Re-replication
  - Re-replicates happens when a chunkserver becomes unavailable
- Rebalancing
  - Periodically rebalance replicas for better disk space and load balancing

Garbage Collection

- Master just logs deletion and rename the file to a hidden name that includes timestamp
- During the masters regular scan, if the timestamp is within recent 3 days (for example) it will not be deleted
  - These files can be read by new name and undeleted by renaming back to the original name
- Periodically check the orphaned chunk and erase them

Stale Replica Detection

- Chunkserver misses mutation to the chunk due to system down
- Master maintains chunk version number to distinguish stale one and up-to-date one
- Increase version number when chunk get lease from master
- Master periodically remove stale replicas

5. 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

6. 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