Google File System

Shashank Shekhar Dubey
Roll no.: CS17S025

IIT Madras

April 27, 2018

Introduction

Agenda

- File System and Distributed File System
- GFS Architecture
- Consistency Model of GFS
- How system interacts
- Conclusion

File System

- Controls storage and retrieval of data
- Groups a piece of data called as 'file'
- Files are isolated and identifiable entities

Distributed File System

- Performance
- Scalability
- Reliability
- Availability

To be noted

The design incorporates the vast experience of Google in handling the data center.

- Component failures are normal and may happen due to
  - Application bugs
  - OS bugs
  - Human errors
  - Failures of disk, memory, connectors, networking, or power supplies.
- System must deal with these problems
  - Constant monitoring
  - Error detection
  - Fault tolerance
  - Automatic recovery

To be noted (Cont...)

- Huge files or billions of small files
  - Moving files is not wise
  - I/O operations and block size should be optimized
- Reading or appending data is more common than overwriting
  - Data analysis
  - Recording data streams

  Hence the optimization is focused on appending.
  eg. Atomic append operation is introduced in GFS.

Design Overview

Usual file system operations supported are:

- Create
- Delete
- Open
- Close
- Read
- Write

New File operations introduced are:

- Snapshots
- Record Append

GFS Architecture

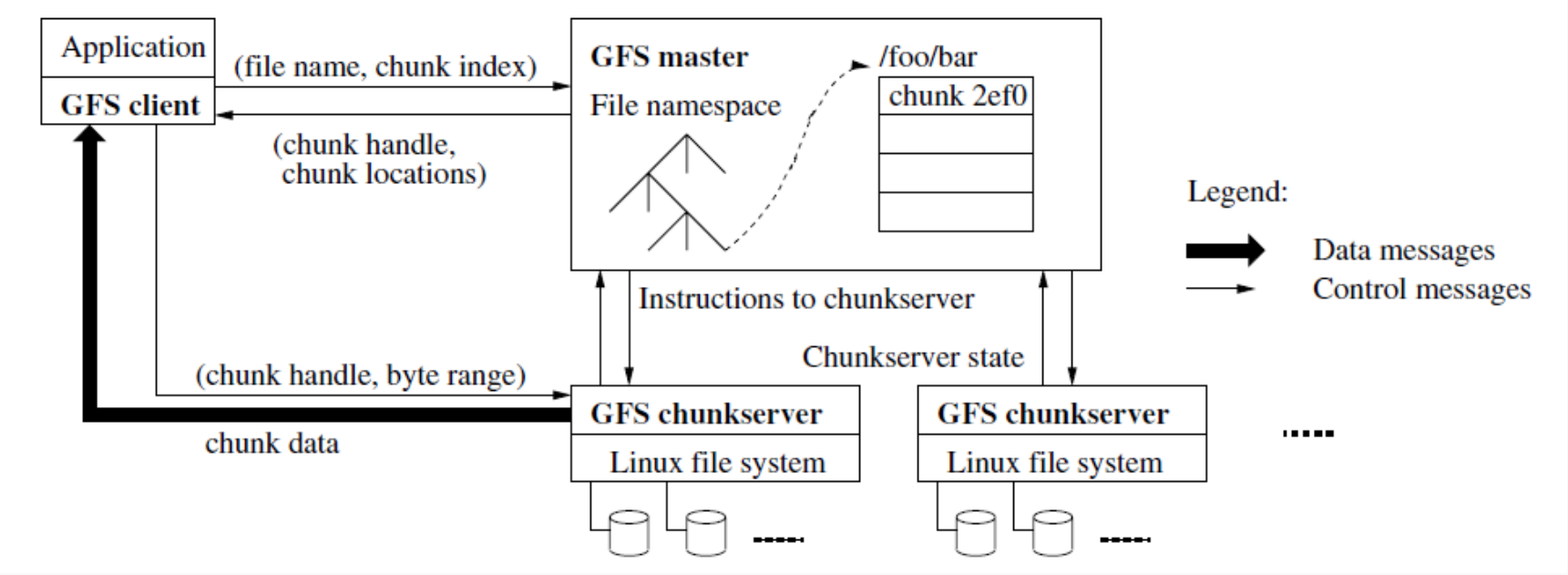

Figure: GFS Architecture

Architecture - Overview

- Major components are GFS Master, ChunkServer, and Client
- In a cluster there can be 1 master, and multiple chunkserver and clients
- Data files are divided into fixed size chunks and stored on chunkserver
- Chunks on the server are identified using globally unique 64bit chunk handle
- Chunks are stored locally on the servers like a linux file
- Each chunk is replicated on multiple chunkservers, by default 3

Master - Overview

- Maintains all the filesystem metadata:
  - Namespaces
  - Access Control
  - Chunk Mapping to files
  - current location of chunk
- Send heartbeat messages to chunkserver
  - to give it instruction and
  - collect its state

Client - Overview

- All the clients implements the filesystem API to communicate in the system
- Client communicates with master for metadata operations
- Data bearing communication goes directly to chunkservers

Single Master

Single master simplifies the design, as it provides global knowledge to the master. For further optimizations:

- Client asks for the r/w file information and caches it
- Further r/w happens directly between client and chunkserver only
- When client asks for the chunk related information, master sends
  - information regarding all the replicas of the chunkserver
  - information related to chunks followed by the requested chunk

  Such measures limits the interaction between Master and Client at almost no extra cost.

Chunk Size

- Maximum size is fixed to 64MB
- Chunks are stored on chunkserver as linux files
- Size is allocated lazily, ie. when needed. Avoids internal fragmentation.
- Why large chunk size?
  - Advantages
    - Communication with master regarding chunk location reduces
    - Metadata at master, stored in-memory is reduced
    - Reduces network overhead, as it maintains single TCP connection for longer time
  - Disadvantages
    - Small files may create hotspots on chunkserver if many clients access the same file

Metadata

- GFS defines 3 types of Metadata
  - File and chunk namespaces, stored in-memory and operational log
  - Mapping from files to chunk, stored in-memory and operational log
  - Location of each chunk replica, stored in-memory

Logs are stored in the local disk and also replicated.
Master asks about chunks location once new chunkserver joins or master startups.

Consistency Model

File namespace mutation is atomic as it is handled by single master.

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

Table: State of the file region after data mutation

- Write mutation, writes data on the file at the offset defined by the application.
- Record append, adds the record at an offset chosen by the GFS.
- Due to concurrent access of the file by multiple processes, some duplication may occur while writing but GFS ensures that data is written atleast once.

*defined but interspersed with inconsistent

How GFS ensures consistency?

After sequence of *successful* mutations, the data region is defined. It is ensured by the following steps:

- applying mutation to a chunk in the same order on all replicas
- using chunk version number to detect any replica that has become stale because, maybe chunkserver was down.
- stale replicas will never be given to clients and get garbage collected.
- Component failure can corrupt or destroy data.
  - GFS does regular handshakes with chunkserver to identify any failure or corruption. Data corruption is identified by checksumming.
  - In case of problem, data is restored from the valid replicas.
  - In case of complete failure of chunk, clear error message is given rather than corrupt data.

System Interactions

All the design is done in a way that masters involvement in any operation is minimum.

We will see how Client, Master and Chunkserver interact to implement:

- Data mutation
- Record append
- Snapshot

Leases and Mutation Order

When multiple clients are accessing a file concurrently, order among replicas is ensured in a following way:

- Client requests master for chunkserver info where its chunk is present.
- Master grants chunk lease to one of the replicas, called *primary*.
- Primary picks the serial order for all mutations to that chunk, and all replicas will follow that order.

Global order is defined by which replica does the master choose to grant the lease, and within that lease, the serial no. assigned by the primary.

Leases and Mutation Order (Cont.)

- Lease has a initial timeout of 60 sec. by default.
- If chunk is being mutated, lease extension can be requested by the chunkserver. Extension does not have limits.
- Extension request are piggybacked on the heartbeat message

Master may also try to revoke a lease before it expires.
Eg. If a file is being renamed, concurrently it would not allow to mutate it.

Write Control

How concurrent access (write mutation) may lead to consistent but undefined state?

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

Table: State of the file region after data mutation

An instance:

If a write by an application is large, GFS client code may break it into smaller write operations. These multiple operations, may interleaved by the operations from other clients.

Once the operation is complete, it will be replicated to other chunkservers in the same way as it is written on the primary. This ensures that every client sees the same copy of the mutation, but this copy may not match to what any single client would have written.

*defined but interspersed with inconsistent

Data Flow

Making Data flow efficient in terms of:

- **Network bandwidth utilization**
  Data is pushed linearly along a chain of chunkservers, instead of any other topology, like tree.
- **Avoid network bottleneck and high latency link**
  Client sends data to the node nearest to it and that node follows the order, until the required chunkserver is found. This avoids the time elapsed due to inter switch links.

Atomic record append

- GFS provides record append mutation operation in which client only specifies the data and GFS choose the offset when appending the data to the file.
- GFS makes sure to do this append operation is done as a continuous sequence of bytes, hence called atomic.

Snapshot

Snapshot operation makes a copy of a file or a directory tree almost instantaneously.

- It can be used to create the copies of huge data sets or to checkpoint the current state.
- When master receives the request to create snapshot, it revokes any outstanding request for lease to that chunk.
- This makes client to request for lease again from the master and master uses this time to create the snapshot.

Master Operation

Master has a wide range of responsibilities:

- It executes all namespace operations
- Manages chunk replicas throughout the system
- Creates new chunks
- Coordinate throughout the system to keep chunks fully replicated
- Balance load across all the chunkservers
- Reclaim unused storage

Namespace management and locking

- Master operations like snapshot revokes the leases on chunks. Such operations delay other operations which is not desirable. Therefore, locks are introduced to allow multiple operations to be active and serialized.
- GFS logically represents its namespace as a lookup table mapping full path name to the metadata.
- Each node in the namespace tree (file/directory) has an associated R/W lock.
- This allows multiple mutations on the same directory concurrently.
- To avoid deadlock due to locking, locks are ordered by the level in the namespace tree and within the same level, lexicographically.

Replica Placement

Purpose of chunk replica is:

- Maximize data reliability and availability,
- Maximize network bandwidth utilization

Chunk replicas are spread across the racks

- **Advantage:** Some replicas will survive even if the entire rack is down.
- **Advantage:** Read will exploit aggregate bandwidth of multiple racks.
- **Disadvantage:** Write traffic has to flow through multiple racks.

Creation, Re-replication, Rebalancing

- Chunk replicas are created for three reasons:
  - chunk creation
  - re-replication
  - rebalancing
- Each chunk that needs to be re-replicated is prioritized as:
  - how far is it from its replication goal.
  - if parent file is live or recently deleted. Live files have higher priority
  - if the chunk is blocking client progress
- Master rebalances the replicas periodically for
  - better disk space utilization
  - load balancing

Garbage Collection

When a file is deleted, the physical space is not immediately reclaimed by the GFS, but does only during normal garbage collection.

- When the file is deleted, it is actually just renamed to a hidden name which includes deletion timestamp.
- During regular file scan if master finds out such a file, then it removes it if the expiry period is passed. Expiry period is by default 3 days, which can be adjusted.
- Until the expiry period file can still be read or undeleted under the newly assigned name.

Master also identifies the orphan chunks, which are unreachable, during its scan and deletes them.

Stale Replica Detection

Chunk replica may become stale if chunkserver fails and misses mutations to the chunks while it is down.

- Stale replicas are identified by chunk version number.
- Whenever master grants new lease to a chunk, it increases the chunk version number and informs the replicas.
- Master and all replicas store the latest chunk version.
- Whenever a chunkserver comes up, from down state, it has to inform master about the chunks it has stored. It also shares the chunk version number. If master finds that the version number provided by this chunkserver is not up to date, then it is considered as stale chunk.

Stale chunks are removed during regular garbage collection

Master also informs the client or chunkserver reading a chunk about the its version number, while directing it to them, so that they can verify before reading that they have the latest version.

Fault Tolerance and Diagnosis

Machines or disks can not be completely trusted and component failure in large data center is quite common. Being always available in such an environment is a challenge.

- High availability
- Data integrity
- Diagnostic tools

High Availability

Strategies to be highly available:

- **Fast recovery**
  Master and chunkservers are designed in a way that they restore their state and start in seconds, independent of how they were terminated. There is no distinction between normal and abnormal termination.
- **Replication**
  - Master's state is replicated for reliability.
  - State here means operation log and checkpoints.
  - If master fails and doesnot come up immediately due to disk failure, may be, GFS starts a new master elsewhere with the same operation log.
  - These are called the shadow masters and provide read only access to the file system.
  - Lag of fraction of seconds can be observed in such scenario.

Data Integrity

- Each chunkserver uses checksum to detect the corruption of stored data.
- Chunks are broken into 64KB blocks and each has 32 bit checksum
- Checksum are metadata, kept in memory, away from user data.

For reads, chunkserver verifies the checksum of required data block, before giving it to the client.

- If the block is corrupted, an error is returned to the requestor and master is informed about the corruption
- The requestor reads from other replicas
- The master clone the chunk from another replica, and then inform the chunkserver to delete its corrupted replica.

Diagnostic tools

- GFS servers generate diagnostic logs about significant events like chunkserver going up and down Or RPC request/replies
- RPC logs contain exact reqeust and response sent
- Interaction history can be recreated by matching requests with replies and collating RPC record on different machines
- Logs are used for load testing and performance analysis

Conclusion

- GFS supports large scale data processing workloads
- Component failure is norm rather than exception
- It optimizes operations like append and read for huge files
- Fault tolerance by constant monitoring, replicating crucial data and fast and automatic recovery
- Detecting corruption using checksum
- Delivers high aggregate throughput to many concurrent readers and writers

References:

The Google File System
Author: Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Conference: SOSP, October 2003

Thankyou