TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A Case Study Report

On

Google file system

Submitted by:

Aayush Lammichhane(075BCT005)

Bishal Katuwal(075BCT028)

BIshant Baniya(075BCT030)

Gobind Prasad Shah(075BCT038)

Submitted to:

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

LALITPUR, NEPAL

August 16, 2022

Introduction:

In 2003, google published a paper called *The Google File System,* with the contributing authors Sanjay Ghemawat, Howard Gobioff and Shun-Tak Leung which was designed to meet the rapidly growing demands of Google's data processing needs, describing their distributed storage system that the google used internally.

This paper and the corresponding architecture was the basis of creation of Hadoop and corresponding to the google file system the component in hadoop is called hadoop file system.

It is essentially a distributed file storage. In any given cluster of google file systems there can be hundreds or thousands of commodity servers. This cluster provides an interface for N number of clients to either read a file or write a file. It is similar to any file system but it is essentially distributed over thousands of servers.

It is used to store huge amounts of data which is difficult to achieve with vertical scaling and because now these files are distributed across the machine it also helps in doing batch processing much faster than a traditional server and that batch processing system is called Mapreduce.

Design consideration while developing Google File System

1. *Commodity hardware:*
   In the early stage of google, they chose to buy off-the-shelf commodity hardware because the commodity hardware was cheap and using a lot of such servers they could easily scale horizontally given the right software layer created on top of it. The main design consideration while using a commercial hardware is that it fails many times due to disk failure, network failure, OS bugs, server crash, or human error. Thus in a cluster of servers at any given time there might be few servers which are always down. The google file system was designed in such a way that it still performs reliably in a fault tolerant manner, encountering all these constant failures.

2. *Large file*
   GFS is optimised to store and read large files typically files in google file system ranges from 100MB to multiple-GBs

3. *File operation*
   GFS is optimised for two kinds of file operations, write to any kinds of files were append-only with no random write in the file. Whereas the read are only sequential reads

Configuration of GFS

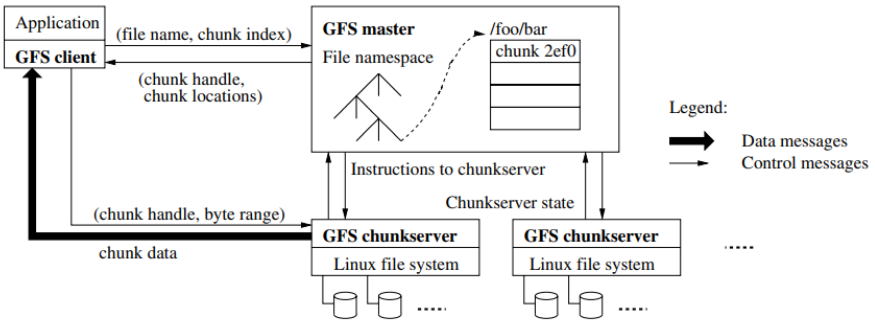


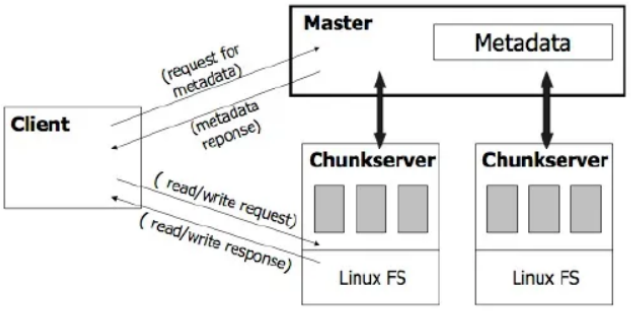
Figure: GFS architecture

- **Chunks**
  The typical file size ranges from 100MB to multiple-GBs and the single file is not stored in a single server. Single file is subdivided into multiple chunks and each chunk of 64MB. These chunks are spread across multiple machines i.e. the commodity hardwares, called chunk server. Each server is not storing an entire file but chunks of a particular file. Each chunk is identified by a globally unique 64 bit ID.
- **Replicas**
  Since the files are stored in commodity hardware, but single commodity hardware is non reliable which can break down at any time and a single copy of a file can be lost forever. Thus the google file system ensures that each chunk of the file has at least three replicas across three different servers. The replica count by default is 3 but is configurable by the client as per the specific requirement. It is difficult for the client application to know which chunk of the file is residing on which chunk server.
  Hence, instead of storing all the data of chunks within the chunk servers or within the client applications there is a separate component called a GFS master which is google file system master server that has all the metadata i.e. the names of all the files in a particular cluster, the number of chunks for each file, the ids of those chunks, and on which servers are those chunks residing.
  GFS master also has the access control details of which client is allowed to access which particular file. Also GFS master has a table which has a file name, chunk IDs which is 64 bit ID and the details of all the replicas of the chunk.

File Access Methods

Read:

Whenever a client wants to read a file, it will ask for the file name, the GFS master from its table will give out the ID of the chunk and all the chunk server IP addresses where that chunk has the replicas. Once the client has those IP addresses, using the particular chunk handle which is the ID, it can directly go to the chunk server and read data directly. Thus the GFS client does not read the data from the GFS master, it only reads the metadata about the file and the actual read- write operation is done only between the client and the chunk server. To further reduce the load from the GFS master the GFS client also caches the particular chunk handle and the chunk locations.



Write

If a GFS client wants to write a file it will ask the master to create a file and the master will give out the locations of all the chunk servers which are relatively free right now, i.e. if there are chunk servers which are only 10-20% full, it will give more priority to those chunk servers, and give the clients three such chunk servers which are free where the client can write the data. Out of those three servers the client will find the closest server and pass all the data to it.

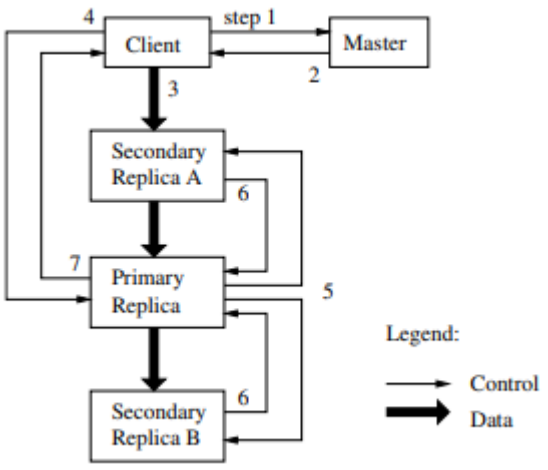


Fig. File write sequence

Replication and Consistency

As seen in the figure, the replica closest to the client sends the data to the replica, and this chunk server will pass the data to another chunk server and so on. Thus the only responsibility of the client is to pass the single replicas and that's how the bandwidth of the system is correctly utilised.

When the master uses those three chunk servers, it says that a particular chunk server is the primary replica.

Now there are three chunk servers all having the data the client wants to write but this data is not immediately stored in the disk but in the LRU cache, and once all the data are written in the cache, the client then sends the request to the primary replicas saying that now you can commit the data on the disk.

The primary replica will coordinate with all other replicas and once it gets a confirmation it will send back the confirmation to the client.

Again there is decoupling of the control between the control instructions and and the data instructions and also the decoupling between the master being used to only get the metadata of a particular file while the interaction of writing to a file is directly done between the client and the chunk server.

- **Heartbeats**

  The chunk servers are off the shelf commodity hardware that could easily go down so it's important for chunk servers to have heartbeat messages passed along with the master so that the master knows that the chunk server is still alive.

- **Ensure chunks replica count**

  In the condition that a particular chunk server died so now the replicas count which was initially three is now reduced to two.

  That particular chunk server will not be able to send the heartbeat message to the GFS master.

  So using the remaining two replica chunk servers the GFS master will ensure that there is one more copy created in any of the chunk servers which are relatively free so that replica count can again be increased to three.

- **Single master for multi-TB cluster**

  Since there is only one master server for entire cluster so even if there are hundreds of clients accessing files, writing, reading or creating the files and there are hundreds of servers, all these operation can still be done using the single master and that is because each chunk is of relatively large size so for each file, the metadata ID X 3 replicas is a very small amount of data that the master needs to store, and the master is not overloaded with data as the client interacts with the master only to get the metadata and once the gets the metadata, it saves the data for a while so it doesn't have to constantly interact with master.

- **Operations log**

  All the operations that occur on the files are stored in an append only log called operations-log.

  Each file operations with the corresponding time stamp and the user details of who performed that operation is stored in the operations log

  The operations log is directly written to disk as well as replicated to some remote machine and only then the acknowledgement is sent to the client

  If the master itself goes down it can backup and read the operations log and create the entire file system namespace along with the chunk IDs to get back to the earlier state of the system.

It is also possible that the operations log becomes too long so there is also an background thread that keeps on checkpointing the operations log

- **Shadow master**
  Since master is the only single component and all the clients talk to it, it becomes a single point of failure i.e. if the master goes down for the clients then the entire GFS cluster goes down.
  So there is another component that does the same operation of the master which runs behind the scene constantly such that if the master goes down the shadow master can take over and perform the operations of the master.

Limitations of the GFS

1. No standard API such as POSIX for programming.
2. The Application / client has the opportunity to get a stale chunk replica, though this probability is low.
3. Some of the performance issues depend on the client and application implementation.
4. If a write by the application is large or straddles a chunk boundary, fragments are added from other clients.