Course Code :	Course Name : DC LAB
Class : BECO	Batch : 1
Roll no : 18CO25	Name : ABDUSSAMAD JAMADAR

AIM: Case Study on Google File System.

Google File System

- Google file system is a scalable distributed file system developed by Google to provide efficient and reliable access to data using large clusters of commodity hardware.
- It is designed to meet the rapidly growing demand of Google's data processing need.
- It provides performance, scalability, reliability and availability of data across distributed system
for handling and processing big data.

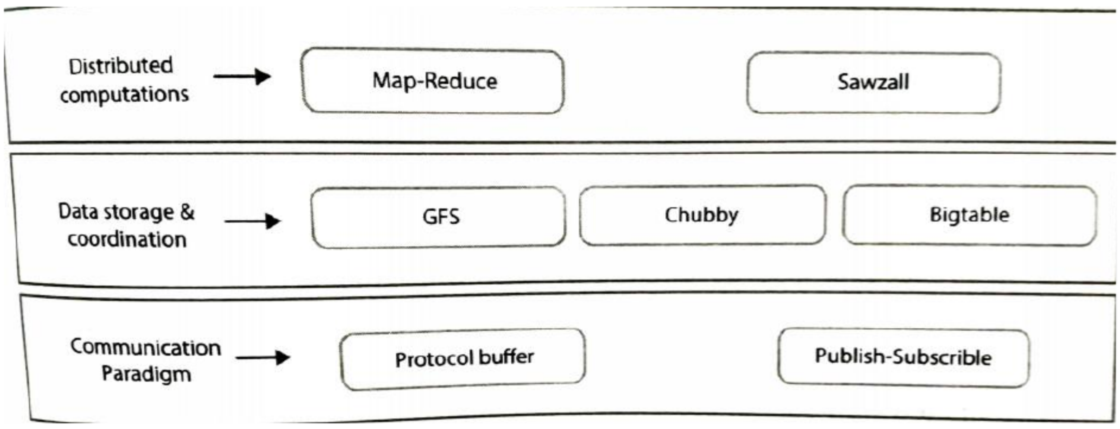
Characteristics of GFS

1. Files are organized hierarchically in directories and identified by path name.
2. It supports all the general operations on files like read, write, open, delete and so on.
3. It provides atomic append operation known as record append.
4. The concurrent writes to the same region are not serializable.
5. It performs two operations: snapshot and record append.

Common goals of GFS

Performance
2. Reliability
3. Automation
4. Fault Tolerance
5. Scalability
6. Availability

Google infrastructure:



It is composed of three layers, namely communication paradigm, data storage and coordination and distributed computations.

These layers provide a set of distributed services to offer communication paradigms and core functionality to developers. The bottommost layer of the infrastructure is communication

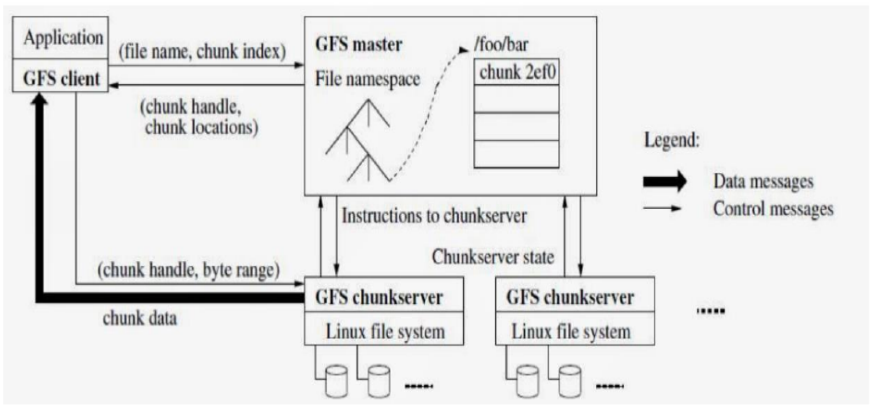
paradigm which has two sub components namely protocol buffer and publish-Subscribe service.
They are used for both remote invocation and indirect communication.

The second layer is data storage and coordination services that provide storage for unstructured
and semi structured data along with services to support coordinated access to the data.

The third layer is distributed computation services, which is used for providing a platform for carrying out parallel and distributed computation over the physical infrastructure.

GFS Architecture:

The basic architecture of Google File System is shown in given figure:



1. Master Node:
- It is responsible for the activities of the system such as managing chunk leases, load balancing and so on.
- It maintains all the file system metadata.
- It contains an operation log that stores namespaces and files to chunk mappings.
- It periodically communicates with chunk server to determine chunk locations and assesses state

of the overall system.
- Each node on the namespace tree has its own read-write lock to manage concurrency.
2. Chunk and Chunk Server
- The files are divided into fixed sized chunks.
- Each chunk has an immutable and globally unique 64-bit chunk handle.
- Chunk server is responsible for storing chunks on local disk as linux files.
- By default, each chunk is replicated 3 times across multiple chunk servers.
- The size of the chunk is 64 MB.
- Due to such a large chunk, it results in space wastage because of internal fragmentation.
- The advantages of large chunk size are as follows:
a) It reduces the client's need to interact with the master. It means reading or writing in a single chunk requires only one request to master.
b) It reduces network overhead by keeping a persistent TCP connection to the chunk server for multiple operations performed by clients.
c) It reduces the size of metadata stored in the master. It enables storage of metadata in memory.
3. Client Node
- Client node is linked with the application that implements GFS API.
- It communicates with the master and the chunk server to read or write data.
- Client communicates with the master to get the metadata.
- For read and write, the client directly communicates with the chunk server.

Operation Log and MetaData
- Operation log is the persistent records of metadata.
- It defines the logical timeline about serialized order of concurrent operations.
- The state is recovered by the master by replaying the operation log.
- The metadata stored in GFS master are as follows:
1. Namespace (directory hierarchy)
2. Access control information per file
3. Mapping from file to chunk
4. Current location of chunks (Chunk servers)

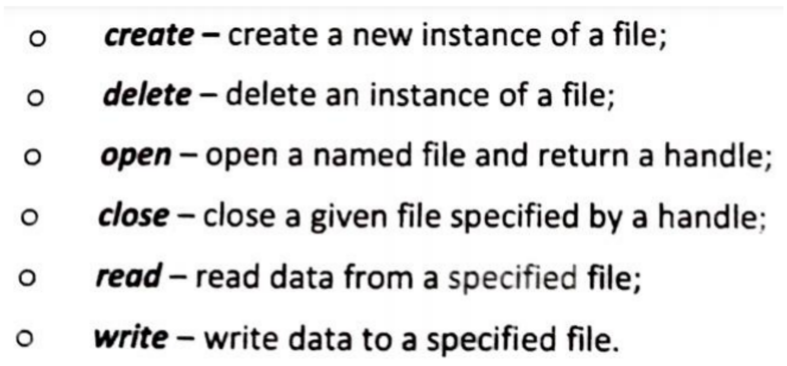
Reasons for Single Master in GFS
- Single master design simplifies the GFS design.
- It enables the master to make sophisticated chunk placement and replication decisions.
- Since, the master interacts with client for sharing metadata only, there is no necessity to have multiple masters.
- All the heavy processes of read and write are handled by the chunk servers. So, lot of chunk servers are necessary for performance of the system.

Manage overloading of the Master

- The read and write operation is completely separated from the master. Instead, these operations are
performed by the chunk servers.
- For reliability, master state is replicated on multiple machines, using the operation logs and checkpoints.
- If the master fails, GFS starts a new master process at any of these replicas.
- Such a replica of the master state is known as shadow master.
- Shadow master is also able to perform read only access to the file system even when the The primary master is down, but it lags the primary master by a few fractions of a second.

Google File System:

The main GFS operations are very similar to those for the flat file service.



- ***create*** **–** create a new instance of a file;
- ***delete*** – delete an instance of a file;
- ***open*** – open a named file and return a handle;
- ***close*** – close a given file specified by a handle;
- ***read*** – read data from a specified file;
- ***write*** – write data to a specified file.