**Google File System**

**Why GFS?**

In today’s world Data is everything. If we take all social networking sites together then in every seconds trillions of data get uploaded on internet. In order to manage this Data a Scalable distributed file system is built for large distributed data-intensive application. The requirement of the file system is generated from the workloads and technology environment of the Google. The authors describe the design decisions and mechanism behind a scalable, reliable and fault tolerant file system which runs on inexpensive hardware.

**Assumptions**

Google File System is designed for our needs while keeping care of some assumptions that offer challenges and opportunities.

A- The system is built from inexpensive components that often fail due to which continuous monitoring, fault detection, tolerance and recover immediately from the system failure are routine tasks.

B- Size of files are always very large (1 MB, 1 GB). Multi-gigabytes files are common and should be managed efficiently.

C- There are two type of reads: large streaming reads and small random reads. The workloads also have many large, sequential writes that append data to files.

D- High sustained bandwidth is more important than low latency.

**Architecture**

The Google File System Architecture is composed by using 3 main components.

1. Master
2. Chunkserver
3. Client

A single master which is tied to multiple Chunkservers and all those multiple chunkservers tied to multiple clients. Files are stored in cluster. Files are divided into 64 MB chunks, each with a globally unique 64bit handle and a version number. Each chunk is stored as a file in the underlying file system and is replicated on multiple chunkservers, which improves bandwidth for read access patterns in a distributed environment. All metadata operations happen at master, data transfer happens directly between client and chunkserver.

Instead of using a directory structure, a lookup table along with prefix compression is used which leads to faster name to chunkserver translation. All metadata is In-Memory. Copy on write mechanism is used based on the observation of frequent append access patterns. Data transfers happen directly on the chunkserver after metadata operations. Overall, all optimization that are needed for the file access inside Google's distributed data processing needs have been done. POSIX compliance is not provided. An atomic append operation is provided, which ensures consistency even with concurrent updates. Control and Data flow is decoupled for maximum network bandwidth utilization. Master maintains Operation log to recover from crashes. Master state is also replicated on a remote site. Lazy garbage collection: deleted files don't immediately get freed, they are renamed and hidden and cleaned up at a later time. Block checksumming is used to ensure data consistency within a chunk.

**System Interaction**

System Interaction basically shows how Master / Chunkserver / Client interacts with Lease / Mutation Order / Data Flow / Snapshot.

**Lease**

1. Minimized management overhead.
2. Picks a serial order of mutation and all replicas follow.
3. 60 Second timeout, can be extended can be revoked.

**Data Flow**

1. Decouple Data Flow and control Flow.
2. Control Flow (Master – Primary - Secondaries)

**Snapshot**

1. Make a copy of a file or a directory tree almost instantaneously.
2. Use Copy on write.

**Master Operation**

The master executes all namespace operations. In addition, it manages chunkreplicas throughout the system: it makes placement decisions, creates new chunks and hence replicas, and coordinates various system-wide activities to keep chunks fully replicated, to balance load across all the chunkservers, and to reclaim unused storage.

Namespace Management

1. No Pre directory data structure.
2. No Support for alias.
3. Lookup table mapping full pathnames to metadata.

Namespace Locking

1. Each node has read/write lock.

Garbage Collection

1. Lazy Reclamation
2. Logs deletion Immediately
3. Regular Scan for orphaned Chunks.

**Fault Tolerance**

One of our greatest challenges in designing the system is dealing with frequent component failures.

High Availability

1. Fast Recovery
2. Chunk Replication
3. Keep each chunk fully replicated as chunk server.

**Design Flaws**

1. The single master with multiple hot replicas will face problems when metadata update operations pressure builds up in cases like small file update. (Note however that metadata read pressure can be alleviated with shadow read only masters. And also when a single master fails, then a replica master can assume mastership almost instantly). As in any centralized system, the server becomes the bottleneck for the system. Since only the master knows all the metadata maps, any downtime of the master leads to downtime of the whole cluster.
2. The paper mentions the problem of hotspots where-in small files which get accommodated in a single chunk are accessed heavily. This problem could occur with large files too where a specific portion of the file, extending to a few 10s of KBs could be very important and be heavily used. This is an artifact of large chunk size where-in the entire portion of the important data would often be present in the same chunk.
3. The consistency guarantees given by the system is not bulletproof: it is up to the client application to interpret the file contents while reading and ensure logical consistency. Lazy reclamation of deleted file space might stop new file creation because of quota restrictions.
4. One general flaw is that this file stem is very specific to Google's data access patterns; it is unclear if any of these techniques can be used for general purpose file systems.

**Conclusion**

1. GFS support Large Scale Data processing using commodity hardware.
2. Reexamine traditional file system assumption.
3. Fault Tolerance.
4. Constant monitoring.
5. Fast and Automatic recovery.
6. High Aggregate throughput.
7. Decouple control and data transfer.

**References**

<http://fr.slideshare.net/tutchiio/gfs-google-file-system>

<http://www.cs.rochester.edu/~sandhya/csc256/seminars/google_fs.pdf>

<https://en.wikipedia.org/wiki/Google_File_System>