GFS AND CHUBBY LOCK SERVICE

GFS was designed to cater to Google's current and anticipated application workloads and technical environment. GFS assumes that failures are central to a system of such scale, therefore it is critical for the system to be able to constantly monitor itself. Detecting, tolerating, and recovering promptly from component failures. The system is also optimized with respect to large files (100MB or larger). Performance optimization is heavily focused on appending operations; this was a design decision that stemmed from continuous observation of Google's applications where random writes were almost non-existent. GFS design also asserts the fact that co-designing the applications and the file system API increase the flexibility of a system. GFS' relaxed consistency model to retain simplicity of the file system. GFS design also caters the fact that for most applications high sustained bandwidth is more important than low latency.

GFS does not implement the standard POSIX API. It has its own interface that has files organized hierarchically in directories, and are identified by path names. It supports traditional operations like create, delete, open, close, read, write, and also introduces support for snapshot and record append operations. Snapshot creates a copy of a file or directory tree almost instantaneously while minimizing any interruptions of ongoing mutations. Snapshot enables the user to quickly create branch copies of huge data sets, or to checkpoint the current state before performing experimental mutations. Record append is an atomic append operation that allows multiple clients to append data to the same file concurrently while guaranteeing the atomicity of each individual client's append.

GFS is made of a single master, multiple chunkservers and is accessed by multiple clients. Files are divided into fixed-size chunks and each is identified by a globally unique 64bit chunk handle assigned by the master at the time of chunk creation. Chunkservers store chunks on local disks like Linux files, and read or write data specified by chunk handle and byte range. Each chunk is replicated on multiple chunkservers to ensure reliability, the default number of replicas is three.

The master maintains three major types of metadata including file and chunk namespace, mapping from files to chunks, and current locations of each chunks' replicas. All metadata is kept in master's memory, while file and chunk namespace and mappings are kept persistent; it does not store chunk location data persistently and this is because chunkserver decides which chunks it does or does not have on its disks. It controls chunk lease management, access control information, garbage collection and chunk migration. The master periodically communicates with chunkservers using HeartBeat messages to give them instructions and collect their state.

Operation log is central to GFS as it is the only persistent record of metadata and also lays out a logical time line that defines the order of concurrent operations. It is replicated across multiple remote machines. The master recovers its state by replaying this log. The master makes a checkpoint of its state whenever the log grows beyond a certain size, which is a compact just a B-tree like form that can be directly mapped into memory and used for namespace lookup without extra parsing, to improve availability and speed up recovery.

GFS is designed to minimize master's involvement in all operations. Leases are used to maintain consistent mutation order across chunk replicas. Master grants chunk lease to one of the chunk replicas called primary, which then picks serial order for mutations to the chunk. This mechanism minimizes overhead at the master. This leads to decoupling of data flow from control flow. Data is pushed linearly along a carefully picked chain of chunkservers in a pipelined fashion which helps to fully utilize each machine's network bandwidth.
GFS is uses Fast Recovery and Replication to ensure high availability of the system. Master and chunkserver are both designed to automatically restore their state regardless of the cause of their termination. Chunks are not only replicated across different machines but on different racks as well. The master state is replicated across multiple remote machines. This provides enhanced reliability in event of catastrophic events.

Data integrity is essential to any file system. Each chunkserver on GFS uses checksums to ensure integrity of stored data. GFS does not guarantee identical replicas therefore, it is essential for each chunkserver to verify its own copy. Chunkservers do not propagate corruptions to other machines as it verifies checksum of data blocks before returning any data to the requestor. As most other operations, checksums are highly optimized for appends in GFS.

Chubby is a lock service that is designed as low volume storage for loosely coupled distributed system of moderately large size. Chubby is expected to help Google's developers to deal with coarse-grained synchronization within their system. GFS and BigTable both use Chubby as the root of their distributed data structures. The design choice of having a lock service was motivated by the fact that it makes its easier to maintain existing program structure and communication patterns.

Chubby has two main components: server and chubby client library. Each of which communicate via RPC. A chubby cell consists of five replicas so as to reduce correlated failure. The replicas then use a distributed consensus protocol to elect a master. The master lease renewed periodically by the replicas as long as it wins the majority. Chubby has a file system similar to UNIX. It consists of a tree of files and directories. The design differs from UNIX in a way that eases distribution. The client also ensures consistent caching for reads. The caching protocol invalidates cached data on change and never updates them. Users are authenticated by a built-in RPC mechanism. Chubby also exposes 64-bit checksum so that clients can easily tell if files differ.

Chubby locks are advisory as they are generally used to protect resources implemented by other resources. Google developers perform conventional error checking and therefore, mandatory locks are almost of no use. Chubby handle is a pointer to an opaque structure that supports various operations. Chubby clients can also subscribe to events when they create a handle. The events are delivered asynchronously via an up-call from the Chubby library. A handle is associated with an instance of a file rather than it's name. Chubby also provides sequencer based checking so that clients can validate the lock.