The Google File System: A Summary

Mariana Ávalos Arce
Distributed Computing

1 Yet Another File System?

The **Google File System** is a distributed file system that has the common goals among other distributed file systems: scalability, reliability and availability. But GFS has to be developed to meet the growing demands of Google's data processing. Therefore, it is a departure from other traditional approaches by what I think is GFS's **main differentiator** in terms of design assumptions: in GFS, component failures are assumed as the **norm**, rather than an **exception**, like in traditional distributed file systems.

GFS has this norm of failure due to the fact that the system is built using thousands of other machines that constitute their distributed network. These machines, as they tend to be quite many, are Linux inexpensive computers, and this leads to component failures and guarantee that its parts are not the best quality, involving errors and failures that happen continuously across the GFS. Therefore, the system must constantly monitor itself, detect, tolerate and repair from errors. These leads up to other big differences and improvements of a distributed FS, which will appear further below.

2 Advantages of GFS

Some of the many advantages of this implementation will be explained in general terms in the following subsections.

2.1 Architecture That Avoids Bottlenecks

One of the main advantages of this file system is that its architecture allows a large scale of data processing without having a *bottleneck* due to constant centralized atomic operations. This is done through the following structure: there is a **master** machine and tons of **chunkservers** machines that store *chunks* or parts of a file that is in this file system. A chunk has a fixed size of 64 MB, which is way larger than usual blocks inside traditional file systems. This avoids a bottleneck of atomic operations when a client requests to handle a file, because the master returns this big chunk where the desired file part is, and avoids constant client-master interaction that would cause delay because of the master's centralized powers. Therefore, this architecture based on big chunks reduces the contact from the client and the traffic of locked operations that the master has with each request.

2.2 GFS Does Not Cache Data

Google's FS does not cache data, as their traffic involves huge file manipulations. Big files like this make caching offer little benefit because of the large data workplace Google has. Therefore, not having to cache on information that is too large, simplifies the model of the system, because there is no computation needed to control cache and its coherence issues. The only cache needed is already implemented by the Linux OS in the servers.

2.3 Replicas

Another big advantage of this file system model is how they implement **replicas** in order to increment **reliability**. GFS has these **chunkservers** that have not only a chunk of a file, but also can have replicas or duplicates of another chunks of other chunkservers. The master has a map of all files to its chunks and also to its replicas that is small in memory, fast to access and improves recovery in case of faults or errors, because the system can switch to a valid replica and continue the work, leaving the faulty replica behind.

2.4 Garbage Collection At 'Nights'

When a client wants to change a file, the master chooses a chunkserver to have the control of the mutations. This permission is called a **lease**. Whenever the master gives a lease to a chunkserver, it increases the chunk version number in itself and in all the chunkservers that contain a replica of the desired chunk. If there is a fault that occurs in any of these machines, the version number is simply not updated. The master then collects garbage such as these stale replicas when it is relatively **free**, simply by eliminating the replicas with a smaller number than the current version. This simplifies the model of the file system and makes it save memory in a reliable way, because the master is not compromised to do it immediately when traffic is heavy and fault likely.

2.5 Atomic Record Appends

GFS has **record appends** instead of file over-writing, and this operation is done atomically: in traditional systems, when a file chunk is modified, the client machine specifies the offset where the new data needs to be written. If it happens concurrently, the offsets of the many machines can differ and cause faulted fragments. Instead, GFS provides the offset atomically and only once to all clients. This avoids the race conditions and the robust task of maintaining a complicated synchronization of a distributed lock manager, which is prompt to errors and data corruption.

3 Fault Tolerance: Mechanisms

Due to the constant dealing with component failures, GFS has a few mechanisms for Fault Tolerance:

3.1 High Availability: Fast Recovery

The master and the chunkservers are designed to be able to restore their state quickly, no matter how the process handled occurred. They restore their state by themselves by handling checkpoints that were constantly made in their metadata file. This makes a machine restore its past state after a fault or simply by routine. Even the master can restore itself from the checkpoints written previously on its operation log, using no more than a few seconds because it restores the checkpoint and its successors after the fault, instead of the whole living pace of the entity.

3.2 Checksumming

A chunk is broken up into 64 KB blocks, and each block has a 32 bit space dedicated to 'checksum' data. Whenever a chunkserver reads from a chunk it compares its checksum with other blocks that overlap, and checks its accuracy before returning any response to the requestor. Therefore, chunkservers do not propagate faults across others. If a checksum does not match with other block's checksum, an error is reported to the requester and the master. This will cause the requestor to read from another replica that has no faults, while the master clones another replica and later collects the faulted one as garbage.

3.3 Constant Diagnostic Logging

For problem isolation, teh GFS always makes extensive and detailed logging when the system goes through backstage operations, that sometimes are even non-repeatable actions inside the system. The servers generate these diagnostic logs that record the significant events across the computations made, recording requests and responses, except the data that goes through them, building a history of interaction that helps trace an error. The performance costs are little, because these logs are made asynchronously while happening.

4 High Availability: How Does It Work?

High Availability is part of how does the GFS implements a Fault Tolerance, due to the fact that the system is expected to fail constantly. Nevertheless, Google maintains the system highly available with two mechanisms: fast recovery and replication.

Fast Recovery means that the main structures (master and chunkservers) are made to restore their last state in seconds, either because of abnormal termination or by routine. Clients and the other parts of the system experience a small 'hiccup' and they reconnect to the restarted entity. Then, as mentioned before, **Chunk Replication** is done because all chunks are replicated several times throughout the system of chunkservers. The master is the one that replicates clones when detecting a stale replica or a faulted replica, in order to always keep the same amount of duplicates of each chunk (by default is three). Additionally, **Master Replication** can also be performed, as the master log is replicated also. This log includes the checkpoints of the master and all replicas have the same log. Just like the chunks, one master is in charge of updating all checkpoints in all replicas. Whenever replication is needed, the master can restart instantly by duplicating the operation log that is constantly updated on the other *shadow* replicas. They provide read-only access and improve availability when the master is down.