Peer review - The Google File System

Fernanda Mora - Maestría en Ciencias en Computación, ITAM

I. Summary

This paper describes the design and performance of the Google File System, a distributed file system built to support Google's large-scale data workloads on commodity hardware. It provides fault tolerance and delivers high aggregate performance to a large number of clients. GFS is widely used within Google in R&D as well as in production but can be applied to other tasks of similar magnitude and cost constrictions.

II. Problem definition

Google shared some baseline design objectives regarding a distributed file system: performance, scalability, reliability and availability. But Google's particular workloads and technological environment restructured them:

- Failures are the norm rather than the exception. The machines used are inexpensive and thus failure prone (some temporarily, some won't recover) and accessed by many clients. Constant monitoring, fault detection/tolerance and automatic recovery are mandatory.
- Efficient handling of huge multi-GB files. Working with billions of small-sized subfiles is not reasonable, hence design assumptions and parameters such as I/O operation and block sizes should be reconsidered.
- Large streaming reads and small random reads. Multiple clients appending data is much more common than overwriting, so it is the focus for performance and atomicity.
- High sustained bandwidth over low latency.

III. Contribution

A. Design overview

GFS was built to satisfy the abovementioned design objectives favoring *simplicity*:

- One single *master* communicating with the clients and maintaining metadata stored in memory: file and chunk namespaces and mapping, and the location of replicas. The first two are also kept persistently in an *operation log* stored at the master and replicated on remote machines, which is batched to reduce overhead and then flushed to disk before responding to a client; this *log* also serves to order concurrent operations and can recover easily. Master makes chunk placement and replication decision based on global knowledge and it just maintains metadata operation. Master doesn't keep info about which chunk server has which chunk replica persistently (so it doesn't have to synchronize with chunk servers) but interchanges regular heartbeats.
- Multiple *clients* and hundreds of *chunkservers* that store file chunks that are replicated as needed (default is three copies) among other chunkservers. The actual read/write is served by chunk server, not by the master. These file chunks were chosen large (64 MB) to reduce the need to communicate with the master (operations on the same file only contact the master once), reduce network overhead (persistent TCP connection) and reduce size of metadata allowing to keep it in memory. They use lazy allocation to avoid fragmentation.

The consistency model in GFS supports highly distributed apps but remains simple. Apps distinguish between a *defined* region (if it is consistent and all write updates are seen) and an *undefined* region (with successful concurrent mutations or with a failed mutation). This is granted using appends, checkpointing and writing self-validating and self-identifying records.

B. System interactions and Master operations

Leases are used to maintain consistent mutation order across replicas. The master grants a lease to one of the replicas -hence reducing overhead on the master-, choosing an order followed by every replica. GFS also provides *atomic record append* and snapshot operation (an instantaneous copy a file).

Master preserves consistency through namespace locking, replica distribution, rebalancing and garbage collection. This improves reliability, availability, and network bandwidth use.

C. Fault tolerance and diagnosis

Log, chunk and master replicas (shadow masters) allow failure recovery achieving availability and reliability. Data integrity is managed with checksum.

D. Measurements and experiences

The first test was under controlled workloads and the other two under real world. Record append was the most common operation. Master efficiently managed the metadata and operations. Most of the results were successful except for network saturation.

IV. Critic

GFS relies too heavily in network bandwidth but experiments showed network saturation, which is extremely important for performance. Master is limited by its memory capacity.

GFS is extremely tailored-made, which questions whether it can really be used in other instances.

It is not clear if GFS's design decisions are optimal for their workloads or simply sufficient.

References

[1] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS operating systems review. Vol. 37. No. 5. ACM, 2003.