The Google File System

Prakhar Srivastav (ps2894)

Summary

The motivation behind the design goals of the authors of GFS was not to create a general purpose filesystem but instead to build a distributed system that could scale to large data which is optimized for Google's specfic workloads. This specific usage and workload gave rise to the following requirements. These were the key goals that GFS set out to solve.

- The need to rely on commodity machines instead of specialized hardware.
- Making the system capable of large files (each of roughly multi-gb).
- Support and optimize a read heavy (specfically streaming reads) workload
- Append-only writes would be dominate and files would rarely be modified
- Allow concurrent access for reads and writes across large number of producers / consumers.
- High sustained bandwidth being more important than low latency.

Hence, the key design decisions that resulted from these requirements were -

- A single master to coordinate with multiple slaves. These slaves were called chunkservers.
- Files would be stored as chunks each of 64 MB. These would reside on these chunkservers.
- To keep the system resistent to failures each chunk would be replicated on multiple machines. The master, would in turn, store the location of each chunk with itself.
- To ensure the master is not a single-point-of-failure, the master uses a write-ahead log (called an *operation log*) to track each mutation done in the files. These logs are then replicated to multiple shadow masters that can be promoted to master in the event of a failure.
- Provide a familiar file system interface. However, the entire POSIX api (like NFS) was not implemented.
- The GFS client would communicate with the master and chunkservers to read or write data on behalf of the application. All data-bearing communication goes directly to the chunkservers, whereas the master would participate only in metadata operations.
- The master keeps the entire metadata in memory. This ensures that the performance is fast and it also allows for background process to do a full scan of the system state.
- Heartbeat messages act as the key method by which the master keeps itself up-to-date on the status of various chunkservers.
- To maintain consistency of data during mutations, the master picks one of the replicas at random and designates it to be a primary copy. The client however sends the data to be changed to all the replicas. The primary replica is responsible for finally deciding the ordering of these events which the other replicas simply follow.
- Lastly, the GFS systems exposes a relaxed consistency model. Concurrent writes leave the region consistent but possibly undefined - ie. all clients see the same data but it may not reflect what any one mutation has writtenConcurrent writes leave the region consistent but possibly undefined - ie. all clients see the same data but it may not reflect what any one mutation has written.

The bottomline of all these design decisions is that the GFS is a highly scalable and highly available file system store. It allows for fast recovery and horizontal scalability to deal with workloads that fill squarely with its assumptions.

As a lesson in distributed system design, the paper goes to show that optimizing for specific use-cases (ie. read dominant, append-only writes, throughput more important over latency etc.) is beneficial in the long run. Large systems must be designed to tolerate frequent component failures and a simple solutions such as a single master can be very effective.