Google File System (GFS)

Aranya Aryaman

January 30, 2025

1 Problem Statement

Traditional distributed file systems were not designed to handle the scale and workload of Google's data processing needs. They assumed reliable hardware, while Google's infrastructure relied on inexpensive commodity machines where failures were frequent. Managing large datasets was inefficient with small block sizes, and existing systems emphasized strong consistency, making them complex and difficult to scale. Additionally, high availability was a key requirement, as Google's services needed continuous access to vast amounts of data despite inevitable failures.

2 Major Differences from Earlier Systems

GFS introduced several innovations that set it apart from traditional file systems. It was designed with failure as a norm, incorporating automatic fault detection and recovery mechanisms. Instead of using small blocks, it stored data in large 64MB chunks, reducing metadata overhead and improving sequential access performance. GFS followed a relaxed consistency model, simplifying operations by allowing temporary inconsistencies while ensuring eventual consistency where necessary. It was optimized for append-heavy workloads rather than random modifications and used a single master for metadata management, avoiding excessive coordination overhead. Additionally, GFS implemented automatic load balancing and replication to maximize performance and reliability.

3 GFS Design Requirements

The system was built to be highly scalable, supporting petabytes of storage across thousands of machines. Given the high failure rate of commodity hardware, GFS prioritized fault tolerance through automatic replication and recovery. Rather than focusing on low-latency operations, it was designed for high sustained throughput, making it suitable for processing large datasets efficiently. The architecture also minimized client-master interactions by allowing clients to communicate directly with chunkservers after obtaining metadata. Another key requirement was supporting concurrent appends, which enabled multiple clients to write to the same file without extensive synchronization.

4 Read and Write Operations

Reading a file in GFS begins with the client translating a file name and byte offset into a chunk index, then requesting the master for the chunk's location. The master responds with the chunk handle and replica locations, after which the client directly contacts a chunkserver to retrieve the data. To improve efficiency, clients cache chunk locations and only interact with the master when necessary. If a replica is unavailable or corrupt, the client retries with another.

Writing involves a similar process, starting with the client requesting chunk location and lease information from the master. The master grants a lease to a primary chunkserver, which serializes writes and forwards them to secondary replicas. The client first pushes data to all replicas and then sends a write request to the primary, which assigns an order and ensures that all replicas execute the write consistently. Once all replicas acknowledge completion, the primary notifies the client. If a failure occurs, the client retries the operation.

5 Impact of GFS

GFS had a profound impact on large-scale computing, serving as the foundation for Google's core data infrastructure, including search indexing and log processing. It inspired modern distributed storage systems, particularly Hadoop's HDFS, which became a key component of the big data ecosystem. By prioritizing scalability, fault tolerance, and efficient data handling, GFS influenced cloud storage architectures and web-scale applications. It also demonstrated that strong consistency was not always necessary, paving the way for eventual consistency models used in distributed databases. The introduction of atomic record append enabled efficient multi-client writes, facilitating real-time data aggregation. Moreover, GFS proved that reliable large-scale storage could be built using inexpensive hardware, a principle that continues to shape cloud computing today.