CyberOtters
CST 334 Online
April 21, 2026

Description of the problem being solved

The Google File System (GFS) was designed to address the limitations of traditional file systems, which were not well suited for Google's large-scale, data-intensive workloads. Conventional file systems assume reliable hardware, handle smaller files, and are optimized for local-computing access patterns, making them inefficient for Google's needs. GFS instead targets environments with extremely large files, often in the range of gigabytes to terabytes, that must be processed across many machines. It also takes into account frequent component failures, since the system is built from thousands of low-cost, commodity hardware nodes where failures are expected and considered normal. Additionally, workloads are dominated by large sequential reads and append operations, not small random writes. Because of this, GFS requires a different optimization strategy, so it was designed as a distributed file system that remains reliable, scalable, and efficient under these challenging conditions.

Why this problem is important

This problem is important because Google's data-intensive services, such as crawling, indexing, and large-scale data analysis, require storage infrastructure that is high-throughput and able to scale across many servers simultaneously. These workloads are imperative for serving Google's complex portfolio of services such as Google Search and YouTube. This means that dependable storage with maximum uptime is critical rather than optional. A failure in the underlying storage layer would have a domino effect, disrupting services that billions of users depend on. Since traditional file systems were not designed with this scale or availability in mind, they could not meet Google's demands for fault tolerance, throughput, or concurrent access across many clients. This gap between what existing systems could offer and what Google actually needed is what motivated the authors to design the Google File System.

Idea proposed by the authors

To solve these problems, the authors designed GFS as a distributed, fault-tolerant file system built specifically for Google's Internet-scale data workloads. GFS assumes that machines will fail often, files will be very large, and most operations will involve reading large amounts of data, or appending new data to the end of files. GFS breaks files into large (64 MB) chunks, stores those chunks across many chunkservers, and keeps multiple replicas of each chunk to improve durability and availability. A single master node coordinates each GFS cluster, while clients communicate directly with chunkservers for the actual data transfer, so that the master does not become a bottleneck. GFS also introduces a relaxed consistency model that allows many clients to append to the same file concurrently. Consequently, GFS demonstrates better fault tolerance, scalability, and higher throughput than a traditional file system.

CyberOtters
CST 334 Online
April 21, 2026

How the authors tested their solution

To test their solution to the problem, the authors used both micro-benchmarks from a small test cluster, as well as real-world usage metrics from production clusters. In their testing, the authors found that GFS performed well with read-heavy workloads. With a single reader, their test GFS cluster reached 10 MB/s (80% of the per-client network limit), and with 16 readers it reached 94 MB/s (75% of the link limit). The authors attribute the drop in per-client efficiency to the increased probability that multiple clients will read simultaneously from the same chunkserver. Write performance was not as strong, with the write rate for one client measured at 6.3 MB/s and the aggregate write rate reaching 35 MB/s for 16 clients (2.2 MB/s per client, half the theoretical limit). The authors attributed poor write performance to their network stack, but mentioned that it was not an issue in their production workloads.

In their production clusters, the authors found that real-world usage tended to be read-heavy, and clusters performed well for these workloads, utilizing most of their available bandwidth. Additionally, the authors found that their single-master design did not impose a bottleneck, as it was easily able to keep up with the amount of requests it received. The authors also found that GFS was able to recover from failures well. In one experiment, they killed a chunkserver holding 600 GB of data, and all affected chunks were restored in 23.2 minutes.

Taken together, the authors demonstrate how a distributed file system can be designed to support enormous, real-world workloads using clusters of commodity machines. GFS can scale to terabytes of storage across many disks while still delivering strong performance to large numbers of clients. A key part of this success is minimizing the master’s role in data transfer, so that it does not become a bottleneck. Just as important, GFS is built around the expectation of failure, with constant monitoring, error detection, fault tolerance, and automatic recovery built into the design. The authors have shown that by combining scalability, efficient client access, and practical reliability mechanisms such as logging and replication, a distributed file system can provide a powerful foundation for large-scale, data-intensive applications.