Name: Sharayu Rasal

Google File System Paper Review: Problem Statement Tackled

The Google File System (GFS) was developed to address Google's massive and growing data processing needs. Traditional distributed file systems assumed reliable hardware and smaller files, but these assumptions broke down at Google's scale. With thousands of commodity servers, frequent failures were inevitable, and workloads involved extremely large files (often multi-GB in size) with mostly sequential writes and streaming reads. The challenge was to design a system that could remain fault-tolerant, scalable, and high-performing while operating on inexpensive hardware.

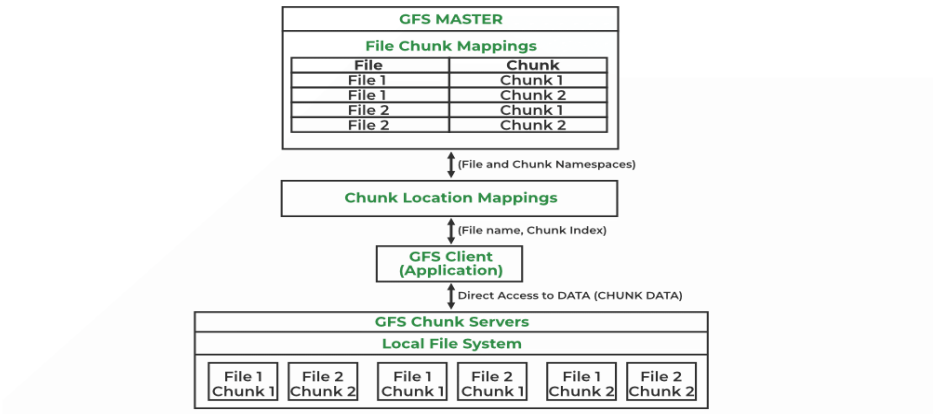
Key Design Principles

- **Failures are normal**: The system is built to constantly monitor itself, detect faults, and recover automatically.
- **Large files dominate**: The system is optimized for handling a modest number of huge files, rather than millions of small ones.
- **Access patterns**: Most operations are sequential streaming reads or large sequential writes, with minimal random modification.
- **Concurrent appends**: Multiple clients frequently append to the same file, so GFS provides atomic record append to handle these safely.
- **Throughput focus**: The design prioritizes sustained high bandwidth over minimizing latency for individual requests.

Architecture & Components

GFS is organized into three main parts: a **single master**, **multiple chunkservers**, and **multiple clients**.

- **Chunking**: Files are divided into fixed-size 64MB chunks, each identified by a unique 64-bit handle. Chunks are stored as Linux files on chunkservers and replicated across three servers by default for reliability.
- **Master**: The master maintains metadata in memory, including namespace, file-to-chunk mappings, and replica locations. Metadata changes are tracked in an operation log, which is replicated across machines to ensure persistence and reliability.
- **Clients**: The Clients contact the master only for metadata. Actual reads and writes are performed directly with chunkservers to avoid bottlenecks.



Data Management

- **Leases**: The master grants a lease to one replica (the primary) to coordinate the order of mutations, while secondary replicas apply them in the same sequence.
- **Record Append**: Ensures atomic append operations from multiple clients, guaranteeing that each append is recorded at least once without collisions.
- **Snapshots**: Provide efficient copy-on-write duplication of files or directories, often used as checkpoints.

Storage Management

- GFS uses **lazy garbage collection** the deleted files are retained under a hidden name for a few days, enabling recovery from accidental deletions before reclaiming space.

Fault Tolerance

- **Fast recovery**: Both masters and chunkservers are designed to restart within seconds.
- **Replication**: Chunks and master metadata are replicated across machines to protect against data loss.
- **Extensive logging**: Diagnostic logs are kept to trace requests and assist in error detection and correction.

Related Work

GFS departs from traditional distributed file systems such as AFS, NFS, and xFS, which were designed under assumptions of more reliable hardware, smaller file sizes, and stricter POSIX compliance. Unlike those systems, GFS relaxes consistency requirements, introduces operations like record append and snapshot, and emphasizes fault tolerance on commodity hardware. These design choices reflect the needs of Google's large-scale data-intensive applications, making GFS unique compared to prior work.

Conclusion / Summary

The Google File System redefined distributed storage by assuming failures as the norm and optimizing for very large, mostly immutable files. Its architecture is centered on a single master with metadata in memory, large chunk sizes, replication, and a fault-tolerant design, which enables Google to handle workloads involving massive datasets with high throughput and reliability. With innovations like atomic record append and snapshot, GFS supported concurrent access at scale while remaining resilient against frequent failures. It became the foundation for later large-scale systems, influencing designs such as Hadoop's HDFS and other distributed storage platforms.