Google file system summary

Traditional file systems weren’t suited for Google’s needs, which involved:

- Handling massive data workloads (web crawling, indexing, and data analytics).
- Large files (often in gigabytes or terabytes).
- Write-heavy operations (e.g., appending logs).
- Frequent hardware failures (due to running on commodity machines).

Design Principles

- Fault Tolerance: Each file is divided into fixed-size chunks (64 MB), and each chunk is replicated across 3 replicas
- Scalability: GFS is designed to scale to thousands of nodes, handling petabytes of data across a distributed cluster.
- High Throughput: GFS prioritizes high sequential read/write performance over low latency, making it suitable for batch processing and large-scale data analysis.
- Simplicity: The design focuses on simplicity to manage complexity and ensure robustness.

Key Features

- Chunk Replication: Each chunk is replicated across multiple servers to ensure fault tolerance. The master monitors chunk health and re-replicates data if a server fails.
- Atomic Appends: Multiple clients can append data to the same file concurrently without locking.
- Consistency Model: GFS uses a relaxed consistency model. Write operations are atomic and consistent, but concurrent writes may result in undefined regions. Applications handle inconsistencies using techniques like checksums or record markers.
- Snapshotting: For backup or versioning

Architecture

- Master Server: A single master node manages metadata (e.g., file names, chunk locations, access controls) and coordinates operations like chunk creation, replication, and rebalancing. The master is periodically backed up to prevent data loss.
- Chunk Servers: Multiple chunk servers store the actual data chunks on local disks. Chunk servers communicate directly with clients for data transfers, reducing the master's bottleneck.
- Clients: Applications interact with GFS via a client library, which communicates with the master for metadata and chunk servers for data.

Impact

GFS laid the foundation for many distributed systems, including Hadoop's HDFS. Its design principles influenced modern distributed storage systems, emphasizing scalability, fault tolerance, and simplicity.