The Day I Lost 84 Centuries

I'm a huge cricket fan. I had collected video highlights of all 84 of Virat Kohli's centuries on my computer's hard drive—my prized collection. One day, I needed more space and discovered the worst: the hard drive was corrupted. All 84 highlights... gone forever. I had no backup.

> Has anyone ever lost a photo, a game save, or a file? It feels terrible, right?

This is the biggest problem with storing something important in just one place. If that one place fails, everything is lost. And what if my collection got even bigger? My single hard drive wouldn't be big enough.

CHAPTER 1

Building a Better System

01

The Problem: Single Point of Failure

My hard drive failed and everything was gone. We need copies!

02

The First Fix: Replication

Make 3 copies of everything. If one machine fails, we still have two safe copies.

03

The New Problem: Three Giant Machines

Finding three empty 1 TB machines at once is nearly impossible—like finding three empty parking lots side-by-side.

04

The Better Fix: Split the File

Chop the 1 TB file into small 64 MB chunks. Sprinkle these chunks across hundreds of machines wherever there's free space.

05

The Final Problem: Finding Everything

If the file is in a million pieces on a thousand machines, how do we ever find it again?

06

The Final Fix: We Need a Map

A single Master keeps track of every chunk's location—the single source of truth.

Congratulations! You've just invented the basic design of the Google File System.

Basic Architecture

Kohli_85_loading.mp4

3 GB File

1 2 3

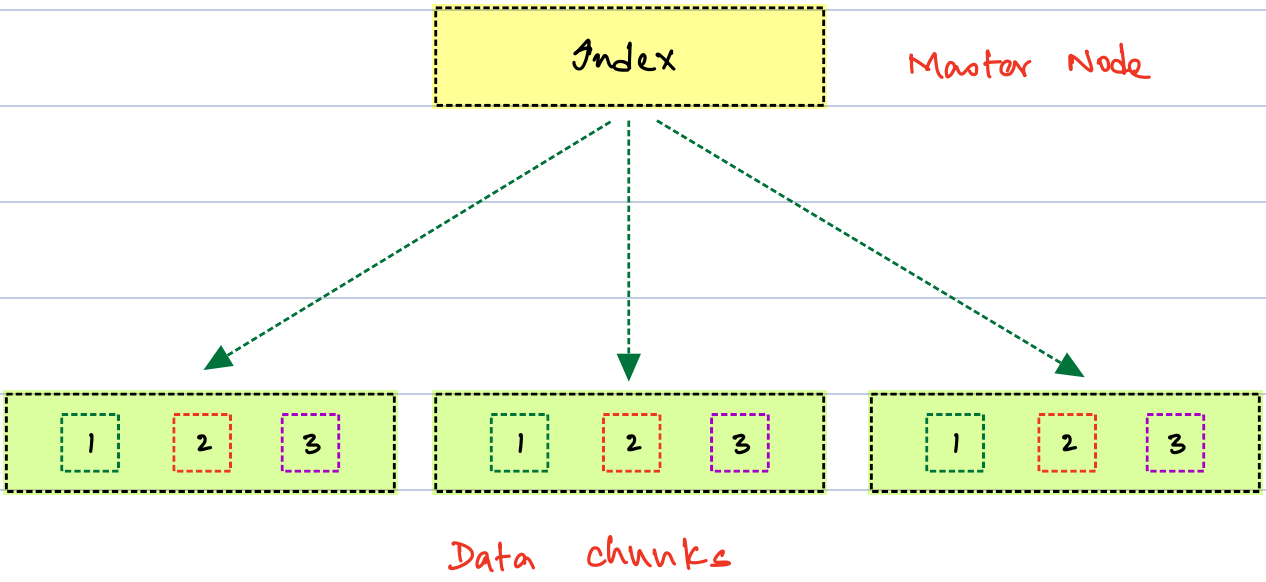
Index

Store these 3 chunks

IP – Mothers Laptop

IP – Laptop

IP – NAS



Congratulations, you have just Invented basic design of GFS

CHAPTER 2

The Rules of the Game: GFS Assumptions

> The perfect system does not exist. Great engineers follow the 80/20 rule: design for the 20% of work you'll do 80% of the time.

1

Hardware Will Fail

GFS expects failure. Hard drives will crash and machines will go offline—this is normal, not an emergency.

2

Files are HUGE

Gigantic files—terabytes in size—much bigger than any single hard drive. Hence, 64MB chunks.

3

Two Main Types of Work

Huge streaming reads (start to finish) and small appending writes (adding to the end). Random writes are rare.

4

Adding New Data is King

Appending is the most important write. The system handles many programs adding data simultaneously without conflicts.

5

Bandwidth Over Speed

Moving data across the network is a bigger bottleneck than CPU speed. Clients get data directly from Chunkservers, not through the Master.

CHAPTER 3

The GFS Master Server: The Brain

Responsibilities

- **Manages All Metadata**
  Stores file/chunk namespaces, file-to-chunk mapping, and chunk replica locations in RAM for fast access.
- **Logs All Changes**
  Writes every metadata-altering operation to an Operation Log on disk for recovery.
- **Monitors Cluster Health**
  Communicates with Chunkservers via Heartbeat messages to track status and chunk locations.
- **Ensures Data Replication**
  If a Chunkserver fails, instructs others to create new replicas to maintain 3 copies.
- **Controls Data Modification**
  Grants chunk leases to designate a primary replica that coordinates write operations.
- **Performs Garbage Collection**
  Renames deleted files, then later removes metadata and instructs Chunkservers to delete chunks.

Major Issues with Single Master Design

Single Point of Failure

If the Master crashes, the entire filesystem becomes unavailable. Failover takes several minutes—no 100% uptime guarantee.

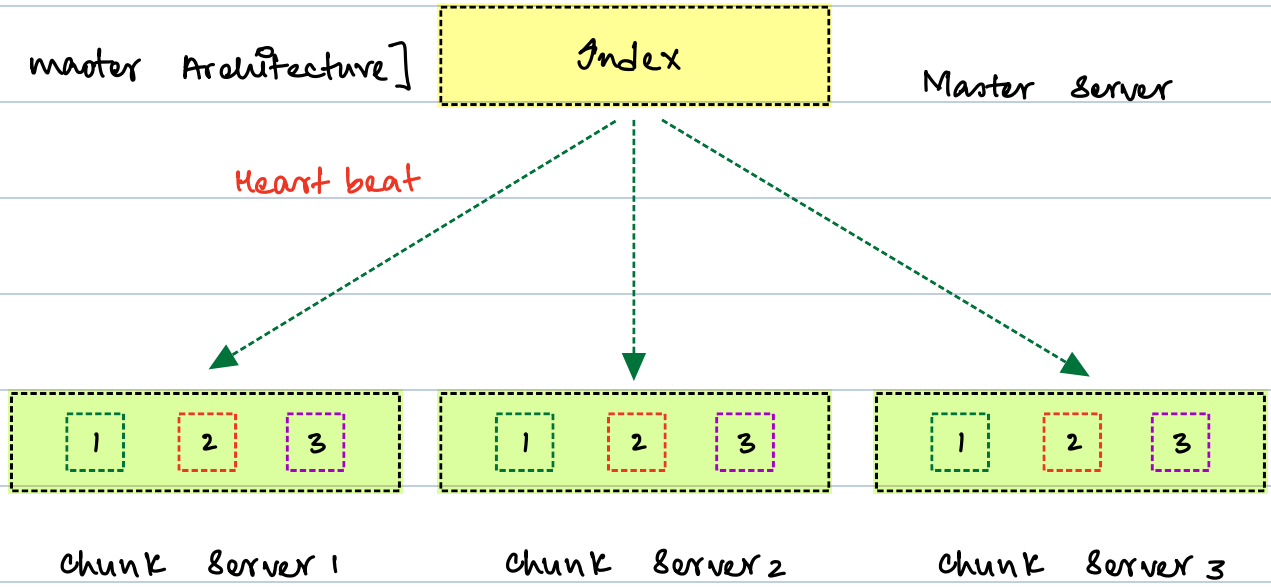
Scalability Bottleneck

Memory limits the number of files/chunks. CPU can be overwhelmed by thousands of simultaneous client operations.

Network Latency

In geographically distributed environments, requests must travel across wide-area networks, adding significant delay.

[Single master Architecture]



Meta data Stored [Held in RAM]

① The entire directory structure & file names

② File to chunk mapping

Kohli_85_loading.mp4 → 1 2 3

③ Chunk Replica locations

File structure / Folder

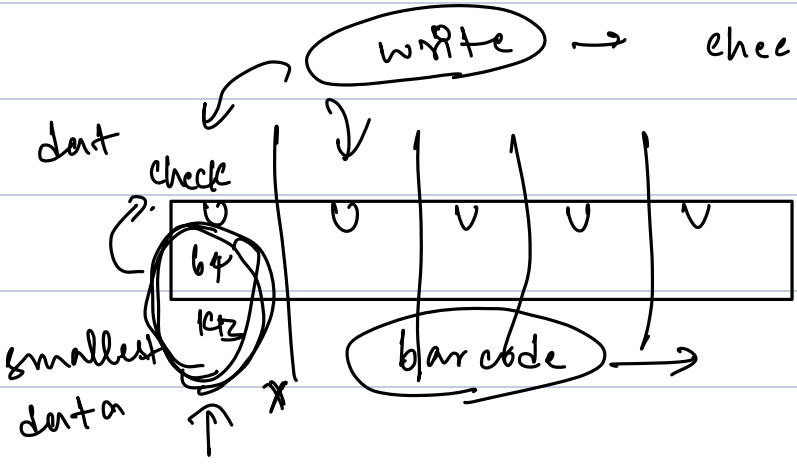
operation.log

Drive

Replicate Drive

Biggest Problem

SPOF



2 other replica

heart

SATA NVME

CHAPTER 4

The GFS Chunkserver: The Data Workhorse

Chunkservers are numerous, simple, and replaceable. Their only job is to store the data.

Store Chunks on Disk

Each chunk is a plain Linux file identified by a unique 64-bit chunk handle.

Serve Read and Write Requests

Responds to requests from clients or other Chunkservers for specific chunk handles and byte ranges.

Maintain Data Integrity

Each 64 MB chunk has 64 KB blocks with checksums. Verifies data before sending; reports corruption.

Communicate with Master

Sends Heartbeat messages to report status and chunk inventory; receives instructions.

Participate in Coordinated Writes

Caches data, then writes to chunk file in precise order specified by the primary replica.

Transfer Data Directly

Sends data directly to clients and other Chunkservers, avoiding the Master as a middleman.

CHAPTER 5

The GFS Read Flow: How a File is Read

Reading is the most common operation, designed to be efficient by minimizing interaction with the Master.

Client Does the Math

Calculates which chunk is needed based on file position and 64 MB chunk size.

Ask Master for Map

Sends filename and chunk index to Master.

Master Replies

Returns chunk handle and list of Chunkserver locations holding replicas.

Client Caches Info

Stores locations in memory to avoid asking Master again.

Read from Chunkserver

Chooses closest Chunkserver and requests data directly.

Key Efficiency: The Master is not involved in actual data transfer. This prevents bottlenecks and saves massive network traffic.

CHAPTER 6

The GFS Write Flow: A Carefully Choreographed Dance

Writing is tricky—all three copies must be updated identically. GFS separates slow data transfer from fast coordinating commands.

Act 1: Getting Permission

Client asks Master for the Primary and secondary replica locations. Master assigns a Primary with a lease.

Act 2: The Data Flow

Client pushes actual data to all three replicas. They store it in temporary buffers—not yet written to final files.

Act 3: The Control Flow

Client sends write command to Primary. Primary assigns serial numbers, applies writes in order, forwards commands to secondaries. Secondaries apply writes in same order.

What Happens on Failure?

If any replica fails, the client receives a failure message. The chunk may be in an inconsistent state. The GFS client retries the entire write operation from the beginning.

File name A

chunk a x 3 chunk c x 3

chunk b x 3

primarily who

— 5 .

① client — file name (appending)

master map LRU Memory

IP1 IP7* IP6

primarily

chunk

② chunck server → client

append : thala for a reason

primary chunk server

thala for a reason	Kpng Kohli	Hitman Rohit

Kpng Kohli	thala for a reason	Hitman Rohit

Hitman Rohit	Kpng Kohli	thala for a reason

CHAPTER 7

The GFS Guarantees: What Can We Trust?

A distributed system can't promise perfection 100% of the time. Instead, it provides clear guarantees about data state after operations.

1

Consistent or Inconsistent

Consistent: All replicas are identical, byte-for-byte. **Inconsistent:** Replicas don't match—a temporary broken state.

2

Successful Writes are Consistent

If your write reports "Success," the region is now Consistent across all replicas.

3

Failed Writes are Inconsistent

If your write reports "Failure," the region is Inconsistent. Client must handle the failure and retry.

4

Namespace Changes are Atomic

Operations like creating or renaming files are handled exclusively by the Master and either complete entirely or not at all.

5

Special Guarantee for Appends

Record append writes data at least once as a complete, unbroken unit, even with many concurrent clients—perfect for log data.

CHAPTER 8

HDFS: The Open-Source Twin Evolves

HDFS started as a faithful implementation of GFS, but the open-source community made significant improvements.

Official Names

- GFS Master → **NameNode**
- GFS Chunkserver → **DataNode**
- GFS Chunks → **Blocks**

High Availability

Two NameNodes: Active and Standby. If Active fails, Standby takes over in seconds—no more single point of failure.

Automatic Failover with ZooKeeper

ZooKeeper manages leader election. Active holds a lock; if it fails, Standby acquires the lock and promotes itself automatically.

Faster Metadata Recovery

Uses FsImage (full snapshot) and EditLog (recent changes). Recovery loads snapshot, then applies recent changes—much faster than replaying entire history.

Larger Block Size

Default 128 MB or 256 MB blocks (vs. GFS 64 MB). More efficient for massive files, reduces metadata overhead.

Robust Write Pipeline

If a DataNode fails mid-write, system removes failed node and continues with remaining nodes—no client restart needed.

HDFS Federation

Multiple NameNodes manage different filesystem parts. Scales horizontally beyond single NameNode memory limits.

CHAPTER 9

The Future is Now: Google Colossus

GFS and HDFS were revolutionary, but final problems remained: the Captain (NameNode) is still a bottleneck, and the map is separate from the data.

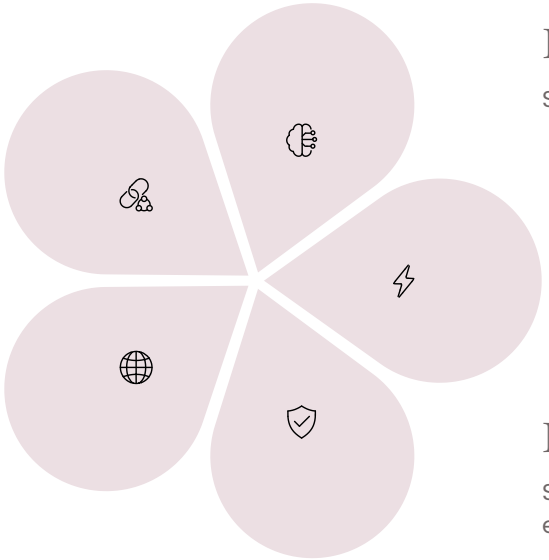
> What if we could get rid of the Captain entirely? What if the data itself was smart enough to know where it belonged?

No Single Master

Metadata is distributed alongside data on chunkservers.

Powers Google Today

Foundation for Drive, Photos, YouTube, and everything you use on Google.



Intelligent Chunkservers

Servers coordinate among themselves to find data.

Direct Access

Go to any server—the system figures out how to get you the data without a central bottleneck.

Massively Resilient

Solves final scaling problems of GFS with unprecedented efficiency.

Next time, we'll explore how Facebook faced a similar challenge storing billions of photos, giving birth to technology that now powers systems like Amazon S3. The world moved from a single "Captain" to a truly decentralized, intelligent "cloud" of data.