



GFS: 大数据存储的奠基者

汇报人: Grissom

时间: 2025. 10





目录

CONTENTS

01 缘起与挑战

02 架构与组件

03 读写与追加

04 一致性与容错

05 性能与扩展





01

缘起与挑战





Google 面临的数据海啸

21世纪初，Google 面临前所未有的数据存储和处理挑战，催生了 GFS 的诞生。



海量数据存储

从网页索引到用户日志，数据量从TB级跃升至PB级。



传统系统瓶颈

单机文件系统在 容量、I/O、可靠性 三方面同时触顶。



GFS 应运而生

以“故障常态、大文件顺序追加”为核心假设，构建分布式存储层。



传统文件系统的四重天花板



强一致性 & 小数据块

POSIX 强一致与 4KB 块带来
跨机锁与元数据爆炸。



容量限制

FAT32 单文件 4GB、ext3 分区
32TB 先撞上限。



可靠性问题

单服务器任一故障即数据丢。



性能瓶颈

百兆网卡 + 磁盘 I/O 成高并发
随机读写瓶颈。

GFS 通过 64MB 大块、三副本、弱一致模型，将三维难题转化为「顺序追加」一维优化。

02

架构与组件



Master - ChunkServer

分工哲学

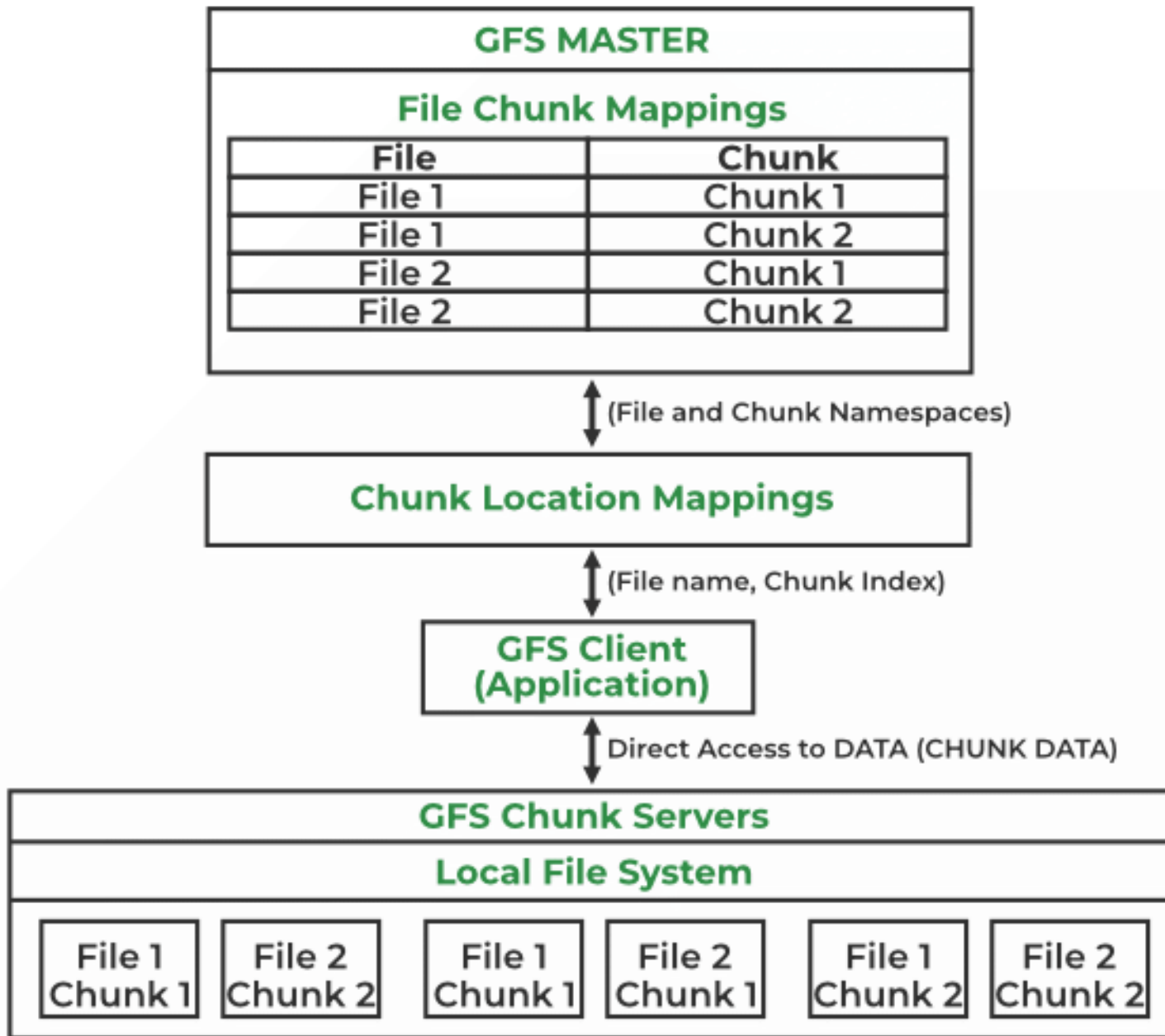
GFS 采用单 Master 全局视图，与作为「磁盘工人」的 ChunkServer 集群，实现控制与数据流的物理分离。

GFS Master

负责命名空间、chunk 租约、负载均衡与垃圾回收。

GFS ChunkServer

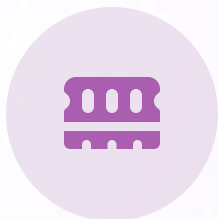
仅存储 64 MB 块文件，并通过心跳向 Master 汇报状态。





单 Master 的瓶颈与解药

单点易成性能与可用性天花板，GFS 通过三招巧妙化解。



元数据全内存 + 持久化

操作日志持久化，影子 Master 提供只读，确保高可用。



Client 侧缓存

缓存 chunk 位置，批量预取，大幅降低 Master 负载。



数据读写旁路

Master 仅处理低频控制，避免成为带宽瓶。



64 MB Chunk 设计得失

大块设计是 GFS 性能与可管理性的核心权衡，旨在优化顺序大 I/O 场景。

+ 优势：减少元数据量，提升顺序读写效率，降低网络交互次数。

- 劣势：小文件空间浪费，热点 chunk 随机读性能下。

⚖ 权衡：以“追加为主”场景对冲，用动态副本与客户端缓存缓解热点。



03

读写与追加

数据读取的三段式路径



1. 元数据获取

Client 计算 chunk 索引，带缓存查询 Master。



2. 副本选择

按网络拓扑选最近 ChunkServer。



3. 数据读取

直发 TCP 请求，顺序读并验证校验和。

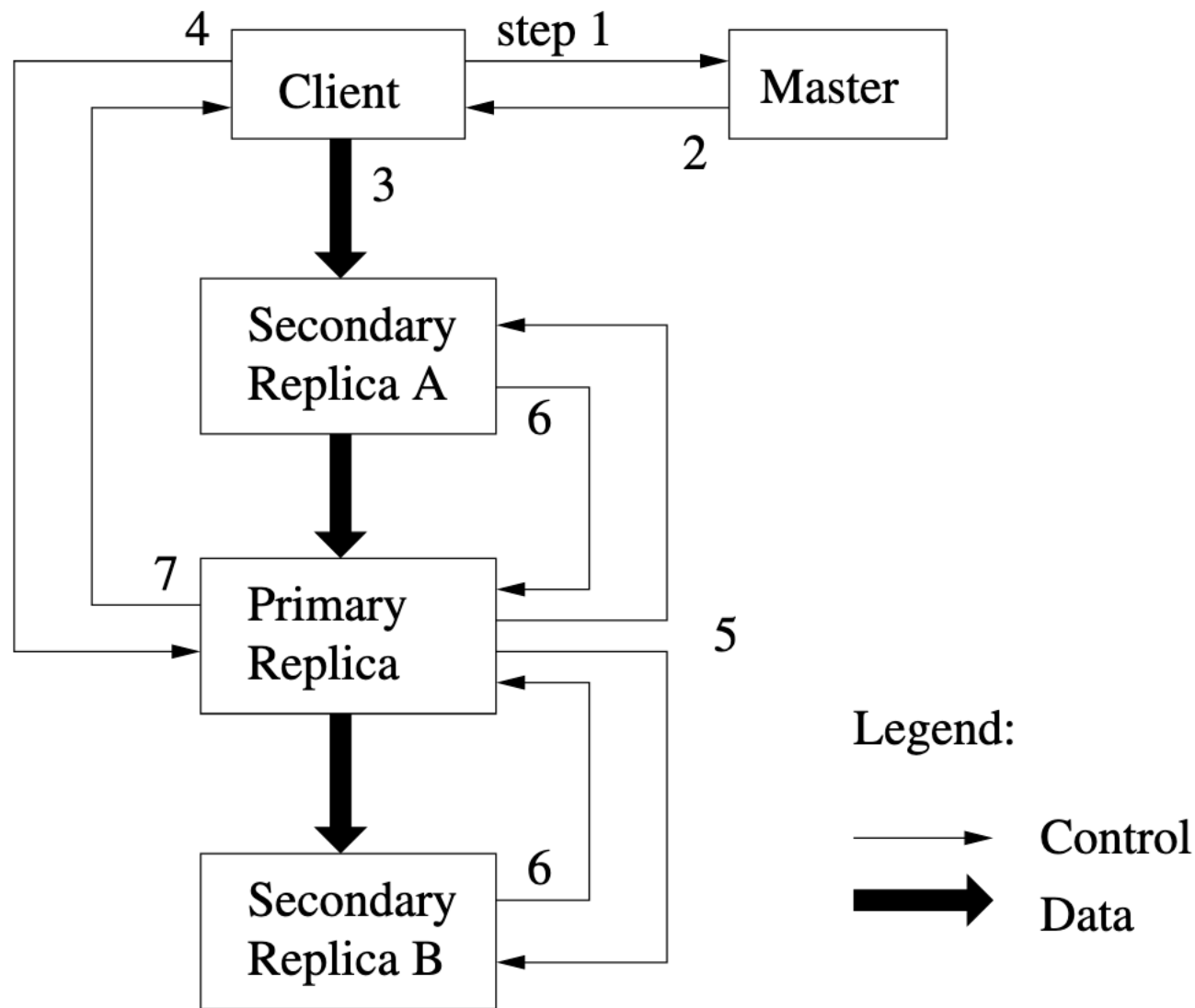
元数据缓存命中95%，同机架RTT < 0.1 ms，聚合读带宽随节点线性增长。



并发写入的租约秩序

通过租约机制，GFS 将并发客户端的随机写转化为副本本地的顺序写，保证全局写入顺序。

1. Client 向 Master 请求 chunk 租约，获取主副本 (Primary)。
2. Client 将数据推送到所有副本，再向 Primary 发写请求。
3. Primary 分配序列号，广播给从副本，按同一顺序落盘。
4. 返回成功后释放租约，确保 **强一致性的写入顺序**。





原子追加的“至少一次”语义

记录追加是 Google 日志场景的核心，提供高并发的原子性保。

- Client 只发数据，Primary 在 chunk 末尾 原子分配偏移 并写入。
- 空间不足时自动填充并返回新 chunk 重试，对 Client 透明。
- 失败重试可能导致记录重复，应用以 校验和 + 唯一 ID 去重。
- 实现「至少一次」到「逻辑一次」的升级，简化失败恢复。

04

一致性与容错



宽松一致性模型解析

一致 (Consistent)

所有副本字节相同，但客户端可能看到不完整写入。

已定义 (Defined)

在一致基础上，客户端能看到完整的、未被破坏的写入内容。

应用适配

应用采用“仅追加 + 自验证记录”模式，处理重复与乱序。

GFS 将一致性责任从 存储后移计算，换取整体吞吐提升。

三副本放置与故障域隔离

默认三副本跨机架分布，可容忍两台服务器或一台机架交换机同时失效。



同节点放置第一个副本（写入节点）。

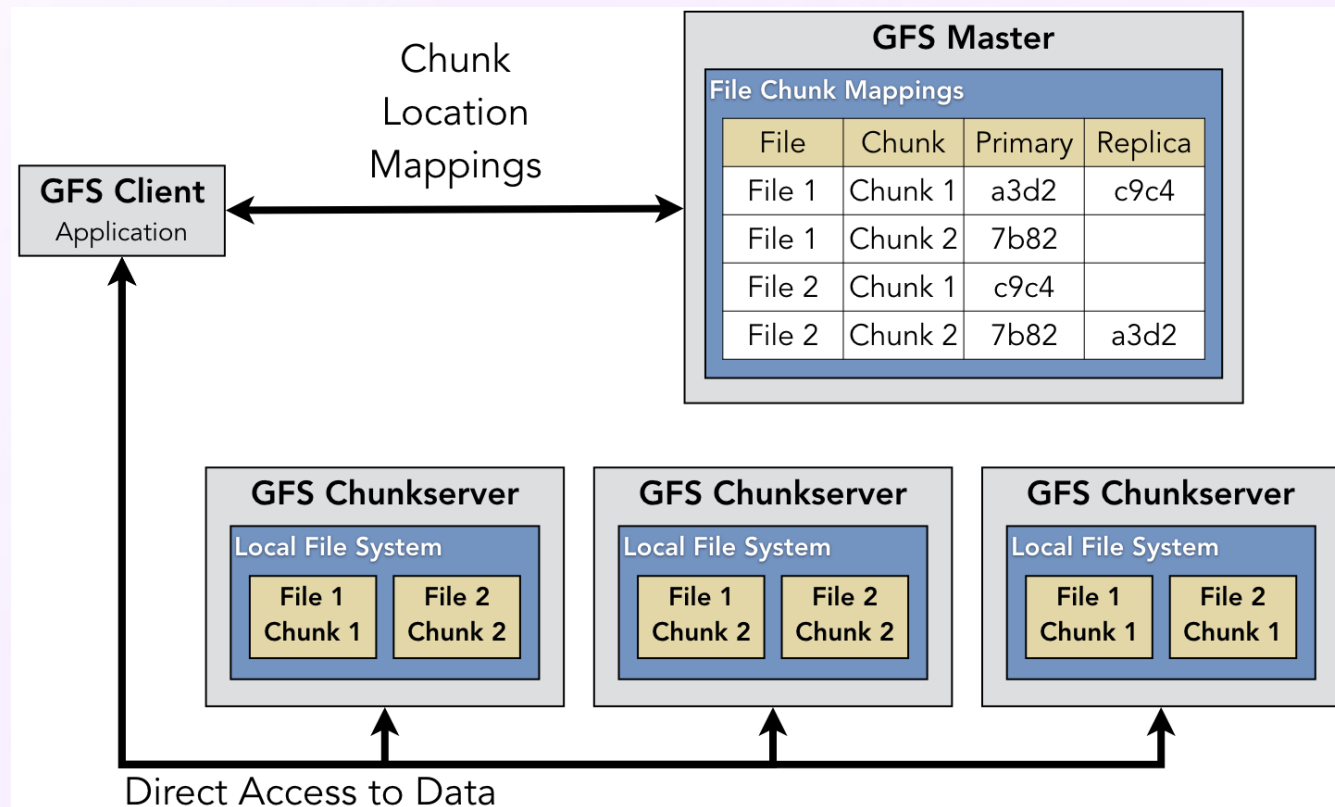


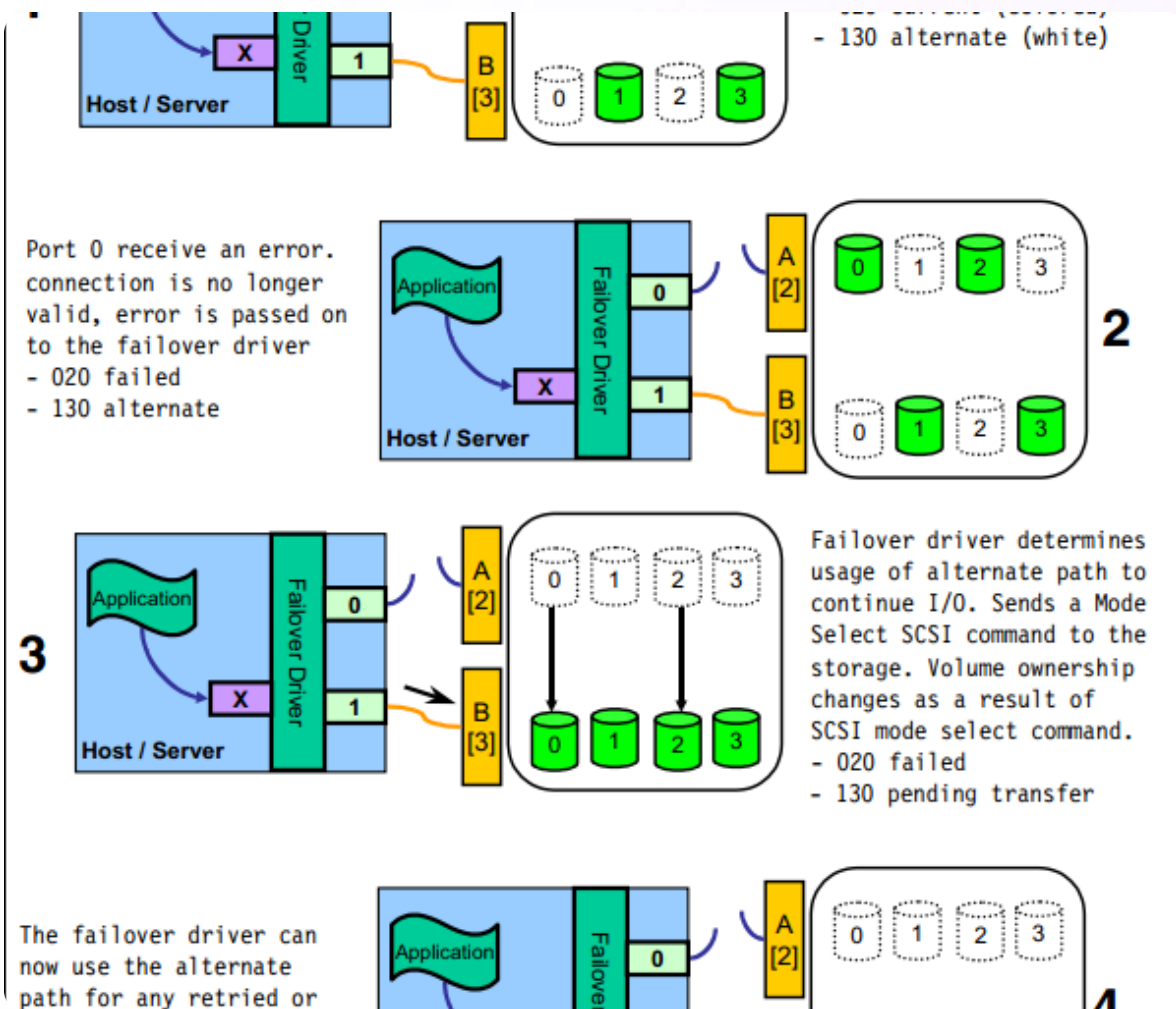
同交换机、不同机架放置第二个副本。



同数据中心、不同交换机放置第三个副本。

Master 动态发起再平衡，30秒内完成检测+复制，年化数据丢失率低于0.005%。





Master 快速故障切换

通过 Primary-Shadow 架构，实现分钟级的故障恢复，保证元数据层零 RPO。

1. 故障检测：Shadow Master 通过心跳超时感知 Primary 宕机。
2. 状态恢复：加载最新检查点，重放操作日志，恢复内存元数据。
3. 服务切换：通知所有 ChunkServer 更新主地址，Client 库自动重连。
4. 快速恢复：整体 RTO < 1 分钟，命名空间与文件到 chunk 映射零丢。

05

性能与扩展



网络拓扑感知的副本选择

读取优化

Client 优先选同节点副本，次选同机架，再次跨机架，显著降低跨核心交换机流量。



同节点



同机架



跨机架

写入优化

采用链式推送，按网络距离构建最小生成树，节省 50 - 70% 骨干带宽。




链式推送 vs 星型推送


拓扑感知使 1 Gbps 网卡集群的聚合读带宽可达 90% 理论上限。




负载均衡与热点消除

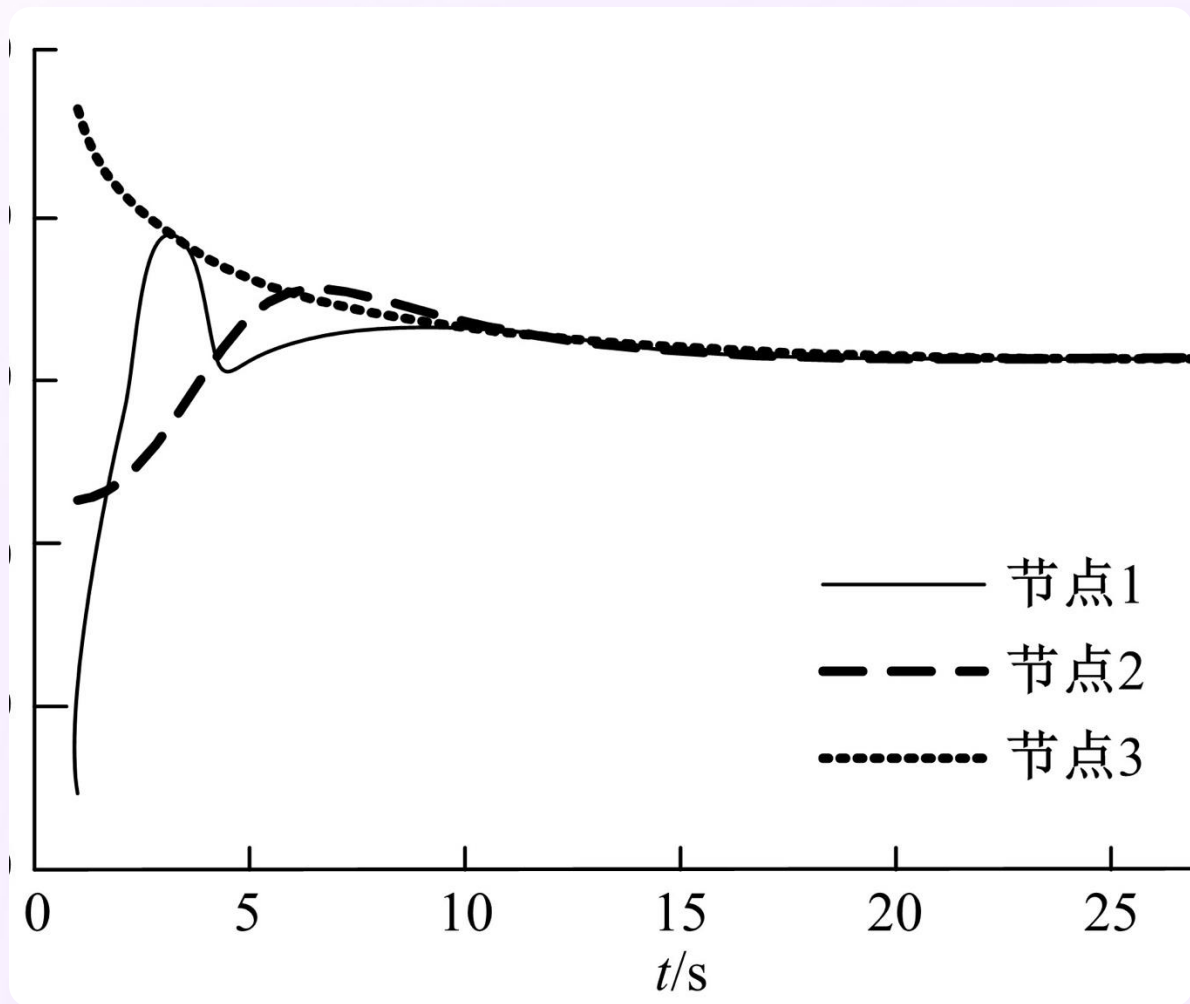
Master 周期扫描集群状态，通过智能迁移和动态副本，实现容量与吞吐双均衡。

 **再平衡触发：**当磁盘使用率或请求队列长度的标准差超阈时触。

 **智能迁移：**高负载节点向低负载节点迁移 chunk，保持故障域约束。

 **热点消除：**对只读热点文件，动态增加临时副本并更新 Client 缓存。

再平衡过程限速 10 MB/s，业务无感知。





水平扩展与天花板



线性扩展

每增一台 ChunkServer，存储 +64 TB、吞吐 +120 MB/s。



Master 内存瓶颈

单 Master 内存随文件数线性增长，约 64 字节/文件，成为首个瓶颈。



实践天花板

Google 实践在 3000 节点、1 PB 级别仍保持线性。

后续通过 分区 Master 与 分布式元数据 把天花板推至 10 PB。

06

局限与演进



单 Master 的固有瓶颈

所有元数据集中导致内存、CPU、网络三热点，催生后续分布式元数据架构。

- **内存瓶颈**：百亿小文件场景下元数据可达百 GB，故障恢复时间随日志长度线性增加。
- **CPU 瓶颈**：全局负载均衡算法复杂度 $O(n^2)$ ，千节点级占单核 30%。
- **网络瓶颈**：跨地域部署时，RTT 放大使租约续期抖动。



小文件与随机写短板



小文件问题

64 MB 块使小文件空间放大 10 - 100 倍，元数据比例高，NameNode 内存被快速耗尽。



随机写问题

覆盖写需锁 chunk 全副本，随机 I/O 抵消大块优势，延迟达数百毫秒。

Google 以 Bigtable 与 Colossus 分别应对，实现场景分层。



从 GFS 到 HDFS 的传承

HDFS 继承了 GFS 的核心思想，并在高可用、一致性和开源生态方面进行了关键改进。

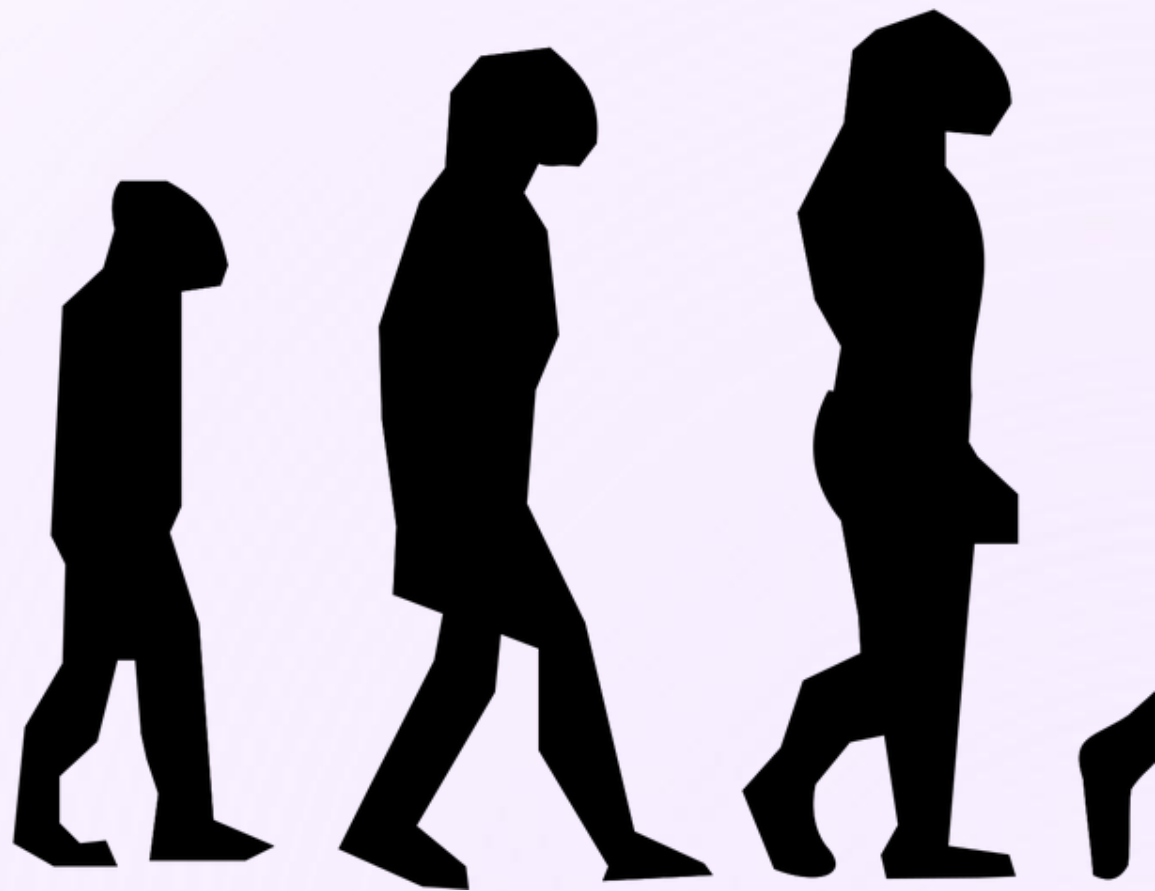
架构传承

Master-Slave、三副本、大块思想。块大小默认 128 MB。

关键改进

NameNode HA、强一致性、开源生态（Hadoop, Spark）。

GFS 的“故障常态、顺序追加、弱一致”三大信条成为大数据领域教材级范式。



07

总结与启示



GFS 的核心设计哲学



故障为导向

接受硬件不可靠，用软件冗余换成本。



应用为驱动

牺牲通用性，换顺序追加极致吞吐。



简单性优先

选择单 Master，降低工程复杂度。



权衡代完美

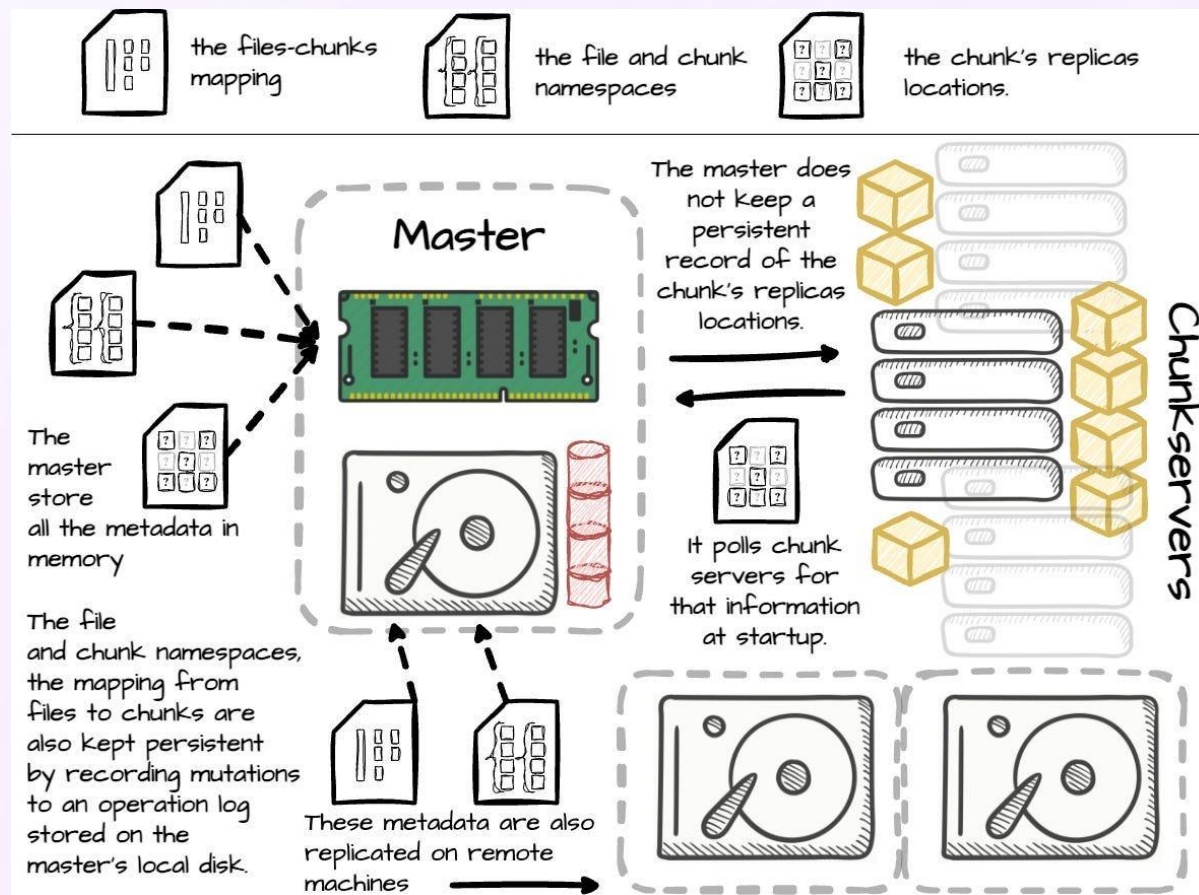
接受弱一致、空间放大，换线性扩展。



对现代系统的持续影响

GFS 证明通过合理取舍，可在廉价组件上构建高可靠、高吞吐、可扩展的存储层，其思想持续指导云与 AI 时代的架构演。

- Colossus, HDFS, Ceph, S3 均沿用三副本与元数据分离。
- 云原生数据库把“租约 + 主副本”搬进 Paxos/Raft。
- 流计算平台继承“仅追加 + 检查点”做 Exactly-Once。
- 对象存储保持最终一致与跨域冗余。





学习 GFS 的方法论价值

GFS 案例教会我们一套「需求→假设→权衡→验证」的闭环设计方法论。

