# Google GFS论文读后感

## GFS的主要需求：

节点失效是常态。系统会构建在大量的普通机器上，这使得节点失效的可能性很高。因此，GFS 必须能有较高的容错性、能够持续地监控自身的状态，同时还要能够顺畅地从节点失效中快速恢复

存储内容以大文件为主。系统需要存储的内容在通常情况下由数量不多的大文件构成，每个文件通常有几百 MB 甚至是几 GB 的大小；系统应当支持小文件，但不需要为其做出优化

主要负载为大容量连续读、小容量随机读以及追加式的连续写

系统应当支持高效且原子的文件追加操作，源于在 Google 的情境中，这些文件多用于生产者-消费者模式或是多路归并

当需要做出取舍时，系统应选择高数据吞吐量而不是低延时

## GFS集群组成：

在存储文件时，GFS 会把文件切分成若干个拥有固定长度的 Chunk（块）并存储。Master 在创建 Chunk 时会为它们赋予一个唯一的 64 位 Handle（句柄），并把它们移交给 Chunk Server，而 Chunk Server 则以普通文件的形式将每个 Chunk 存储在自己的本地磁盘上。为了确保 Chunk 的可用性，GFS 会把每个 Chunk 备份成若干个 Replica 分配到其他 Chunk Server 上。

GFS 的 Master 负责维护整个集群的元数据，包括集群的 Namespace（命名空间，即文件元数据）以及 Chunk Lease 管理、无用 Chunk 回收等系统级操作。Chunk Server 除了保存 Chunk 以外也会周期地和 Master 通过心跳信号进行通信，Master 也借此得以收集每个 Chunk Server 当前的状态，并向其发送指令。

鉴于整个集群只有一个 Master，客户端在和 GFS 集群通信时，首先会从 Master 处获取 GFS 的元数据，而实际文件的数据传输则会与 Chunk Server 直接进行，以避免 Master 成为整个系统的数据传输瓶颈；除此以外，客户端也会在一定时间内缓存 Master 返回的集群元数据。

## GFS的元数据：

GFS 集群的所有元数据都会保存在 Master 的内存中。鉴于整个集群只会有一个 Master，这也使得元数据的管理变得更为简单。GFS 集群的元数据主要包括以下三类信息：

文件与 Chunk 的 Namespace

文件与 Chunk 之间的映射关系

每个 Chunk Replica 所在的位置

元数据保存在 Master 的内存中使得 Master 要对元数据做出变更变得极为容易；同时，这也使得 Master 能够更加高效地扫描集群的元数据，以唤起 Chunk 回收、Chunk 均衡等系统级管理操作。唯一的不足在于这使得整个集群所能拥有的 Chunk 数量受限于 Master 的内存大小，不过从论文的内容来看，这样的瓶颈在 Google 中从来没有被触及过，源于对于一个 64MB 大小的 Chunk，Master 只需要维持不到 64 字节的元数据。况且，相比于增加代码的复杂度，提高 Master 内存容量的成本要小得多。

为了保证元数据的可用性，Master 在对元数据做任何操作前对会用**先写日志**的形式将操作进行记录，日志写入完成后再进行实际操作，而这些日志也会被备份到多个机器上进行保存。不过，Chunk Replica 的位置不会被持久化到日志中，而是由 Master 在启动时询问各个 Chunk Server 其当前所有的 Replica。这可以省去 Master 与 Chunk Server 同步数据的成本，同时进一步简化 Master 日志持久化的工作。这样的设计也是合情合理的，毕竟 Chunk Server 当前实际持有哪些 Replica 也应由 Chunk Server 自己说了算。

## 数据的一致性：

用户在使用 GFS 这类数据存储系统时，首先应当了解其所能提供的数据一致性，而作为学习者我们也应先理解 GFS 对外呈现的数据一致性功能。

首先，命名空间完全由单节点 Master 管理在其内存中，这部分数据的修改可以通过让 Master 为其添加互斥锁来解决并发修改的问题，因此命名空间的数据修改是可以确保完全原子的。

文件的数据修改则相对复杂。在讲述接下来的内容前，首先我们先明确，在文件的某一部分被修改后，它可能进入以下三种状态的其中之一：

客户端读取不同的 Replica 时可能会读取到不同的内容，那这部分文件是不一致的。

所有客户端无论读取哪个 Replica 都会读取到相同的内容，那这部分文件就是一致的。

所有客户端都能看到上一次修改的所有完整内容，且这部分文件是一致的，那么我们说这部分文件是确定的。