精读完经典论文《The Google File System》后，我收获颇丰。下面对该论文进行一个总结。

Google作为一个庞大的数据工厂， 如何高效可靠地存储如此大规模的数据成为一个很棘手的问题 。纵观Google的内部应用，数据访问有以下特点：

1. 数据集庞大，数据总量和单个文件都比较大，如应用常常产生数GB大小的单个文件；
2. 数据访问特点多为顺序访问，比较常见的场景是数据分析，应用程序会顺序遍历数据文件，产生顺序读行为；
3. 多客户端并发追加场景很常见，极少有随机写行为；
4. 一次写入，多次读取，例如互联网上的网页存储。

GFS是Google针对其数据访问模式而设计的分布式存储系统，也是本文的主要贡献，它主要具备以下特点：

- 高度扩展性，单个GFS集群文件系统可扩展至几千个节点，数PB的容量；
- 高可靠性，GFS运行在普通x86服务器上，但GFS通过多副本等机制保证可靠性；
- 满足POSIX语义，应用程序无需任何修改即可使用GFS。

以下是GFS的架构图：

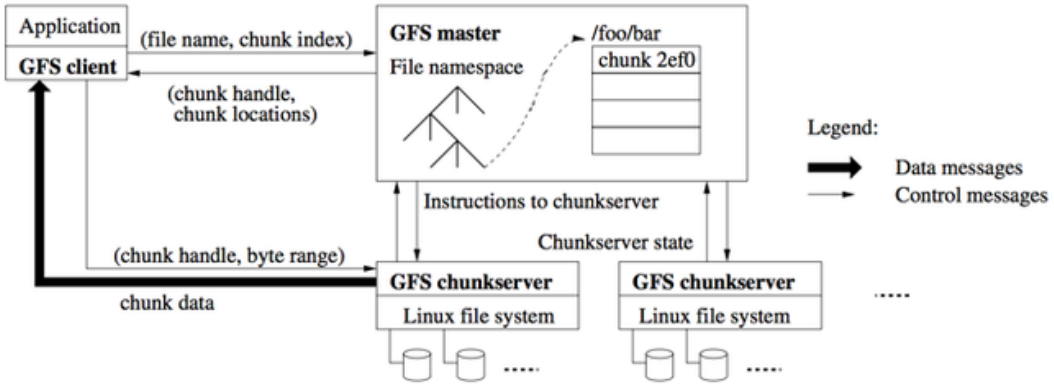


**Figure 1: GFS Architecture**

GFS的核心问题在于定义了**数据一致性**，它一共以下几种一致性：

- **defined**：状态已定义，从客户端角度来看，客户端完全了解已写入集群的数据，例如，客户端串行写入且成功，此时的状态是defined
- **consistent**：客户端来看chunk多副本的数据完全一致，但不一定defined，一般发生在多客户端并发更新时
- **unconsistent**：多副本数据不一致
- **undefined**：数据未定义

GFS在chunk多副本之间选择出一个主副本，由主副本来协调客户端的写入，保证多副本之间维持一个全局统一的更新顺序，GFS使用了租约。

租约（Lease）是由GFS中心节点Master分配给chunk的某个副本的锁。持有租约的副本方可处理客户端的更新请求，客户端更新数据前会从Master获取该chunk持有租约的副本并向该副本发送更新请求。

租约本质上是一种有时间限制的锁：租约的持有者（chunk的某个副本）需要定期向Master申请续约。如果超过租约的期限，那么该租约会被强制收回并重新分配给其他副本。

租约中有一个问题值得思考：假如副本A1、A2、A3（A1是主副本），在写的过程中A3掉线，此时还能继续写么？

可能解决方案：

- 客户端在重试N次后依然写失败，启动数据恢复，在数据恢复过程中该chunk不可写入，此时需要由应用去处理不可写情况（GFS做法）；
- master对此种情况时分配一个新的chunk写入，原不可写chunk后续无法被再写入。
- 降低要求，写入A1、A2即可，后面等master数据恢复将副本3再恢复出来。

GFS的关键流程如下：

**数据写入**

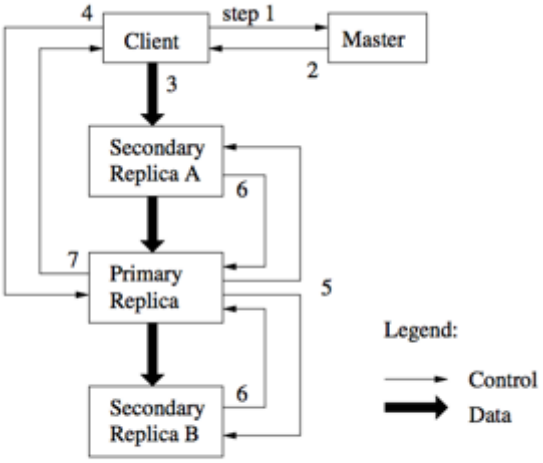


**Figure 2: Write Control and Data Flow**

1. 客户端向Master查询待写入的chunk的副本信息，
2. Master返回副本列表，第一项为主副本，即当前持有租约的副本；
3. 客户端向多副本推送待写入数据，这里的推送是指将数据发送至chunk多副本，chunkserver会缓存这些数据，此时数据并不落盘；
4. 客户端向主副本发起Sync请求；
5. 主副本将数据写入本地的同时通知其他副本将数据写入各自节点，此时数据方才落盘；
6. 主副本等待所有从副本的sync响应；
7. 主副本给客户端返回写入成功响应

客户端的数据吸入被拆成了数据推送和sync两个子命令，这是因为：

- 数据推送过程，客户端可以根据网络拓扑情况进行推送路径优化：客户端可以选择距离自己最近的副本推送数据，然后再由该副本选择下一个距离自己最近的副本进行数据推送，直到数据被扩散至所有副本，由于该过程仅仅是将数据推送给chunkserver并保存在内存缓冲区中，因此，无需保证数据推送顺序；
- sync是将上面推送的数据落盘，需要保证多副本上数据写入序列的一致性，否则大家各人执行各人的，会出现副本之间数据不一致，该指令必须由主副本来确定数据更新顺序然后将该顺序通知给其他从副本。

**chunk副本位置**

GFS中chunk以多副本存储，以提高数据可靠性。因此，副本位置的选取是一个比较关键的问题，一个好的副本位置定义算法满足下面特性：

1. 保证足够的可靠性，例如，不能将所有副本存放在同一个磁盘或者物理机器上；
2. 保证写入高效性，多副本位置尽量靠近，降低写入延迟，提高读写性能

GFS在论文中说明了创建chunk时副本位置的选择算法：

1. 选择存储空间利用率最低的节点和磁盘；
2. 选择最近一段时间内新建chunk数量较少的节点和磁盘；

3. 将多个副本分散在不同的rack上。

1和3比较容易理解，2是为了保证一个节点/磁盘不会被频繁新建chunk（新建完接下来就是数据写入了），否则很容易沦为热点，导致磁盘IO和网络带宽被占满，影响效率。

**Snapshot**

Snapshot是对系统当前状态进行的一次拍照。用户可以在任意时刻回滚到快照的状态。GFS使用COW技术实现Snapshot。

COW原理是如果被Snapshot的文件有更新操作时，就将文件的要被更新的chunk复制一份，然后对复制的chunk进行更新，而原来的chunk作为快照数据被保留，以后要恢复到该快照时，直接将该chunk读出即可。

当GFS的Master节点收到Snapshot请求时：

1. 回收Snapshot请求覆盖的文件chunks上的租约，这样，接下来客户端要对文件修改时，就必须向Master申请，而此时master就可以对chunk进行复制；
2. Master在日志中记录本次Snapshot操作，然后在内存中执行Snapshot动作，具体是将被Snapshot的文件或目录的元数据复制一份，被复制出的文件与原始文件指向相同的chunk；
3. 假如客户端申请更新被Snapshot的文件内容，那么找到需要更新的Chunk，向其多个副本发送拷贝命令，在其本地创建出Chunk的副本Chunk'，之所以本地创建是因为可以避免跨节点之间的数据拷贝，节省网络带宽；
4. 客户端收到Master的响应后，表示该Chunk已经COW结束，接下来客户端的更新流程与正常的没有区别。

GFS作为Google三驾马车之一，为google的强势发展打下了基础。许多技术并不首创的，但是将技术结合使用在GFS，却让Google获益颇多。不论GFS是否适应现在的发展，都对当时的公司产生了深渊影响。 以上就是对论文的一些总结和感想。

最后再附一些关于GFS的一些问题与解答，以此加深理解。

> Q：为什么原子记录追加操作是至少一次（At Least Once），而不是确定一次（Exactly Once）？

要让追加操作做到确定一次是不容易的，因为如此一来 Primary 会需要保存一些状态信息以检测重复的数据，而这些信息也需要复制到其他服务器上，以确保 Primary 失效时这些信息不会丢失。

> Q：应用怎么知道 Chunk 中哪些是填充数据或者重复数据？

要想检测填充数据，应用可以在每个有效记录之前加上一个魔数（Magic Number）进行标记，或者用校验和保证数据的有效性。应用可通过在记录中添加唯一 ID 来检测重复数据，这样应用在读入数据时就可以利用已经读入的 ID 来排除重复的数据了。GFS 本身提供了 library 来支撑这些典型的用例。

> Q：考虑到原子记录追加操作会把数据写入到文件的一个不可预知的偏移值中，客户端该怎么找到它们的数据？

追加操作（以及 GFS 本身）主要是面向那些会完整读取文件的应用的。这些应用会读取所有的记录，所以它们并不需要提前知道记录的位置。例如，一个文件中可能包含若干个并行的网络爬虫获取的所有链接 URL。这些 URL 在文件中的偏移值是不重要的，应用只会想要完整读取所有 URL。

> Q：如果一个应用使用了标准的 POSIX 文件 API，为了使用 GFS 它会需要做出修改吗？

答案是需要的，不过 GFS 并不是设计给已有的应用的，它主要面向的是新开发的应用，如 MapReduce 程序。

> Q：GFS 是怎么确定最近的 Replica 的位置的？

论文中有提到 GFS 是基于保存 Replica 的服务器的 IP 地址来判断距离的。在 2003 年的时候，Google 分配 IP 地址的方式应该确保了如果两个服务器的 IP 地址在 IP 地址空间中较为接近，那么它们在机房中的位置也会较为接近。

> Q：Google 现在还在使用 GFS 吗?

Google 仍然在使用 GFS，而且是作为其他如 BigTable 等存储系统的后端。由于工作负载的扩大以及技术的革新，GFS 的设计在这些年里无疑已经经过大量调整了，但我并不了解其细节。HDFS 是公众可用的对 GFS 的设计的一种效仿，很多公司都在使用它。

> Q：Master 不会成为性能瓶颈吗?

确实有这个可能，GFS 的设计者也花了很多心思来避免这个问题。例如，Master 会把它的状态保存在内存中以快速地进行响应。从实验数据来看，对于大文件读取（GFS 主要针对的负载类型），Master 不是瓶颈所在；对于小文件操作以及目录操作，Master 的性能也还跟得上。

> Q：GFS 为了性能和简洁而牺牲了正确性，这样的选择有多合理呢?

这是分布式系统领域的老问题了。保证强一致性通常需要更加复杂且需要机器间进行更多通信的协议。通过利用某些类型的应用可以容忍较为松懈的一致性的事实，人们就能够设计出拥有良好性能以及足够的一致性的系统。例如，GFS 对 MapReduce 应用做出了特殊优化，这些应用需要的是对大文件的高读取效率，还能够容忍文件中存在数据空洞、重复记录或是不一致的读取结果；另一方面，GFS 则不适用于存储银行账号的存款信息。

> Q：如果 Master 失效了会怎样?

GFS 集群中会有持有 Master 状态完整备份的 Replica Master；通过论文中没有提到的某个机制，GFS 会在 Master 失效时切换到其中一个 Replica。有可能这会需要一个人类管理者的介入来指定一个新的 Master。无论如何，我们都可以确定集群中潜伏着一个故障单点，理论上能够让集群无法从 Master 失效中进行自动恢复。