架构师之路

笔记本:	架构
创建时间:	2018/12/7 23:07
标签:	架构
URL:	https://mp.weixin.qq.com/s?__biz=MjM5ODYxMDA5OQ==&mid=2651961812&idx=1&sn=592e3cc7...

> 搞架构的人，Google的架构论文是必看的，但好像大家都不愿意去啃英文论文。故把自己的读书笔记，加入自己的思考，分享给大家。

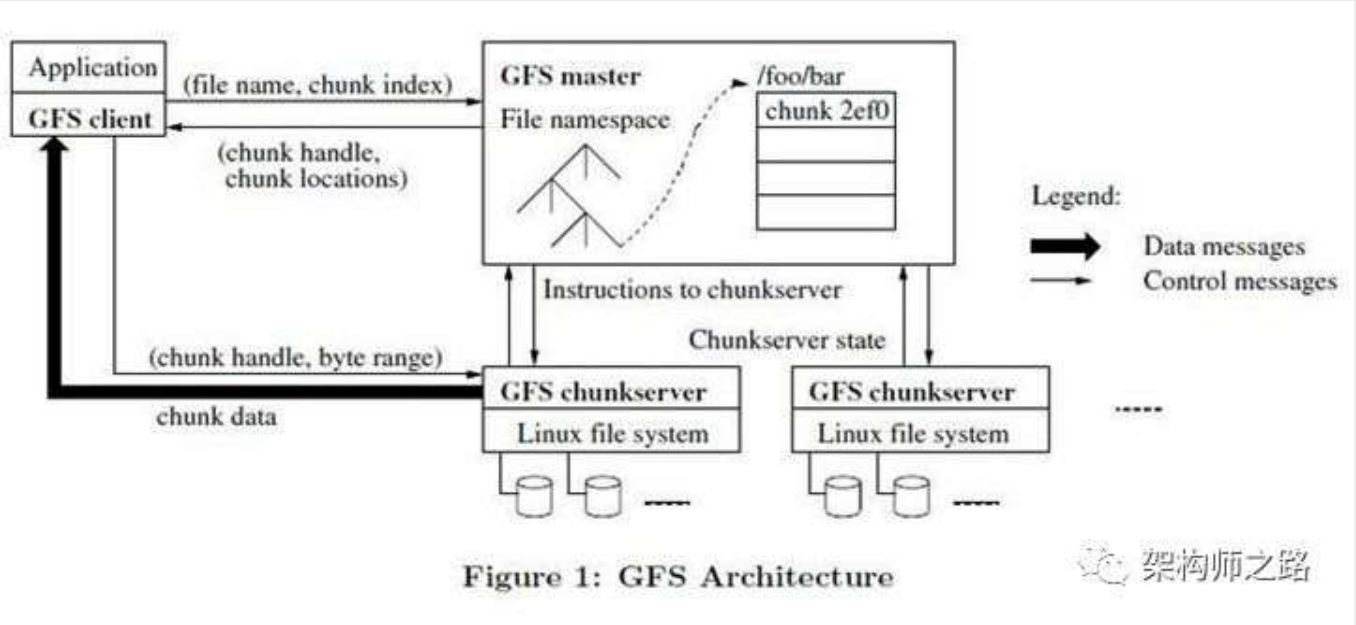
第一篇，GFS（Google File System）架构启示。

GFS是什么?
Google早期研发的分布式文件系统。
画外音：与分布式文件系统对应的，是单机文件系统，Windows和Linux操作系统都有文件系统。

GFS的设计目标是什么?
主要有四个目标：
(1) **高可用**（availability）；
(2) **高可靠**（reliability）；
(3) **高性能**（performance）；
(4) **可扩展**（scalability）；
画外音：WaCaLe，这些词都被架构师用烂了，简单解释一下。
*高可用，是指7*24提供服务，任何一台机器挂了或者磁盘坏了，服务不终止，文件不丢失；*
高可靠，是指争取的输入，得到正确的输出，读取a文件，不会得到b文件；
高性能，是指吞吐量很牛逼，每秒响应几十万请求；
可扩展，是指加机器，就能提升性能，就能存更多文件；
额，希望这个解释是通俗的。

GFS对外提供什么接口?
文件创建，删除，打开，关闭，读，写，快照。
画外音：
除了快照，接口和单机文件系统差不多。
快照其实是快速文件目录树的拷贝，并不是所有文件的快照。
GFS能够成为分布式架构的经典案例，原因之一，就是接口简单，但反映的架构理念不简单。

GFS的系统架构如何?
系统里只有文件客户端，主服务器，存储服务器三个角色。



如上图：

(1) **客户端**（GFS client），是以库的形式提供的，提供的就是对外要用的接口；

(2) **主服务器**（GFS master），是单点，存储文件信息，目录信息，文件服务器信息，那个文件存在哪些文件服务器上等元数据；

(3) **存储服务器**（GFS chunk-server），是集群，存储文件；

画外音：角色简单，但反映的架构理念不简单。

为什么要设计单点master?

单点master意味着有一个节点可以避免分布式锁，可以拥有全局视野，能够统一调度与监控，系统整体复杂度降低很多。

画外音：锁可以降级成本地锁，分布式调度可以降级为单点调度。

更具体的：

(1) master拥有所有文件目录结构，要操作某个文件，必须获得相应的锁；

画外音：一般情况下，不会对同一个网页进行并发写操作，应用场景决定锁冲突其实不大；

(2) master拥有全局视野，能够避免死锁；

(3) master知道chunk-server的信息，能够很容易的做chunk-server监控，负载均衡；

(4) master知道所有文件的副本分布信息，能够很容易的做文件大小的负载均衡；

画外音：负载均衡分为请求量的均衡，文件存储的容量均衡。

GFS的高可用是怎么保证的?

高可用又分为服务高可用，文件存储高可用，均通过“冗余+自动故障转移”的思路是实现。

(1) **master高可用**：冗余了一台影子master，平时不工作，master挂了工作，以保证master的高可用；

画外音：master资源利用率只有50%。

(2) **chunk-server高可用**：本身是集群，冗余服务；

画外音：当有chunk-server挂掉，master能检测到，并且知道哪些文件存储在chunk-server上，就可以启动新的实例，并复制相关文件。

(3) **文件存储高可用**：每一份文件会存三份，冗余文件；

GFS的高性能是怎么保证的?

多个**chunk-server**可以通过线性扩展提升处理能力和存储空间，GFS的潜在瓶颈是单点master，所以GFS要想达到**超高性能，主要架构优化思路**在于，“提升master性能，减少与master交互”。

(1) 只存储元数据，不存储文件数据，不让**磁盘容量**成为master瓶颈；
(2) 元数据会存储在磁盘和内存里，不让**磁盘IO**成为master瓶颈；
(3) 元数据大小内存完全能装得下，不让**内存容量**成为master瓶颈；
(4) 所有数据流，数据缓存，都不走master，不让**带宽**成为master瓶颈；
(5) 元数据可以缓存在客户端，每次从客户端本地缓存访问元数据，只有元数据不准确的时候，才会访问master，不让**CPU**成为成为master瓶颈；

当然，chunk-server虽然有多个，也会通过一些手段提升chunk-server的性能，例如：
(1) 文件块使用64M，避免太多碎片降低性能；
(2) 使用追加写，而不是随机写，提升性能；
(3) 使用TCP长连接，提升性能；

GFS如何保证系统可靠性？
保证元数据与文件数据的可靠性，GFS使用了很多非常经典的手段。
(1) 元数据的变更，会先写日志，以确保不会丢失；
画外音：日志也会冗余，具备高可用。
(2) master会轮询探测chunk-server的存活性，保证有chunk-server失效时，chunk-server的状态是准确的；
画外音：文件会存多份，短时间内chunk-server挂掉是不影响的。
(3) 元数据的修改是原子的，由master控制，master必须保证元数据修改的顺序性；
(4) 文件的正确性，通过checksum保证；
(5) 监控，快速发现问题；

读操作的核心流程？
文件读取是最高频的操作。
(1) client读本地缓存，看文件在哪些chunk-server上；
(2) 如果client本地缓存miss，询问master文件所在位置，并更新本地缓存；
(3) 从一个chunk-server里读文件，如果读取到，就返回；

写操作的核心流程？
写操作会复杂很多。

为了保证数据高可用，数据必须在多个chunk-server上写入多个副本，首先要解决的问题是，**如何保证多个chunk-server上的数据是一致的呢？**
想想一个MySQL集群的多个MySQL实例，是如何保证多个实例的数据一致性的。bingo！确定一个主实例，串行化所有写操作，然后在其他实例重放相同的操作序列，以保证多个实例数据的一致性。

GFS也采用了类似的策略，一个文件冗余3份，存在3个chunk-server上，如下图步骤1-7：

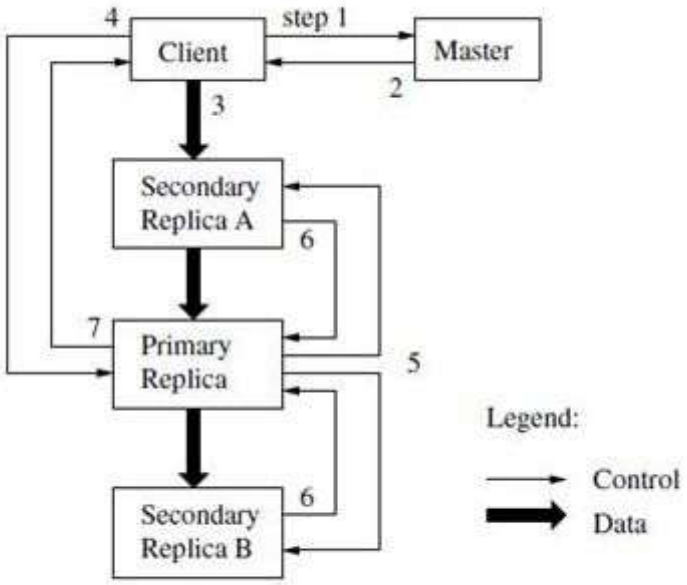


Figure 2: Write Control and Data Flow

(1) client访问master，要发起文件写操作；
画外音：假设client本地缓存未生效；
(2) master返回数据存储在ABC三个实例上，并且告之其中一个实例是主chunk-server；
(3) client将数据流传递给所有chunk-server；
(4) client将控制流产地给主chunk-server；
(5) 主chunk-server进行本地操作串行化，并将序列化后的命令发送给其他chunk-server；
(6) 其他chunk-server按照相同的控制流对数据进行操作，并将结果告诉主chunk-server；
(7) 主chunk-server收到其他所有chunk-server的成果执行结果后，将结果返回client；
画外音：MySQL的主库是写瓶颈，GFS不会出现这样的问题，每个文件的主chunk-server是不同的，所以每个实例的写请求也是均衡的。

这里需要说明的是，GFS对于写操作，执行的是最**保守的策略**，必须所有chunk写成功，才会返回client写成功（写吞吐会降低）；这样的好处是，读操作只要一个chunk读取成功，就能返回读成功（读吞吐会提升）。
画外音：这也符合R+W>N的定理，N=3份副本，W=3写3个副本才算成功，R=1读1个副本就算成功。R+W>N定理未来再详述。

之所以这么设计，和文件操作"读多写少"的特性有关的，Google抓取的网页，更新较少，读取较多，这也是一个设计折衷的典型。
画外音：任何脱离业务的架构设计都是耍流氓。

除此之外，这里还有一个"数据流与控制流分离"的设计准则：
(1) 控制流数据量小，client直接与主chunk-server交互；
(2) 数据流数据量大，client选择"最近的路径"发送数据；
画外音：所谓"最近"，可以通过IP的相似度计算得到。

总结
GFS的架构，体现了很多经典的设计实践：

- 简化系统角色，单点master降低系统复杂度
- 不管是文件还是服务，均通过"冗余+故障自动转移"保证**高可用**

- 由于存在单点master，GFS将“降低与单点master的交互”作为**性能优化核心**
- 通过写日志，原子修改，checksum，快速监控快速恢复等方式保证**可靠性与完整性**
- 通过串行化保证多个副本数据的**一致性**
- 控制流与数据流分离，提高性能

画外音：GFS还有一些优化细节也挺有意思，文章未能穷尽。

架构师之路-分享可落地的技术文章

希望大家对GFS的架构，和设计方法有了初步的了解，希望大家有收获。**有问必回。**

谢转。