**Google三驾马车读后感**

1. **GFS （Google 文件系统）**

GFS是一个可扩展的[分布式文件系统](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F/1250388)，用于大型的、分布式的、对大量数据进行访问的应用。它运行于廉价的普通硬件上，并提供容错功能。它可以给大量的用户提供总体性能较高的服务。

GFS的新颖之处并不在于它采用了多么令人惊讶的新技术，而在于它采用廉价的商用计算机集群构建分布式文件系统，在降低成本的同时经受了实际应用的考验。

GFS主要包括master和chunk server两部分组成。一个文件被打散成了多个chunk，每个chunk会在多个chunk server上保存作为冗余备份。客户端向master请求某个文件的某个chunk的数据，master返回这个chunk所属的chunk server。客户端再请求对应的chunk server操作数据。

其中，master机器只有一台，在内存中存储：

1. File namespace: 可以理解为文件目录结构；

2. 每个文件所包含的chunk；

3. 每一个chunk存放在哪些chunk server上面(不在master进行持久化，master启动时询问所有的chunk server，chunk server加入时也会主动注册)。

这样的设计巧妙的解耦了元数据和数据。对于元数据，只有master知道。因此，master必须好好维护这些信息，防止数据丢失或者错误，因此写operation log就变得非常重要。而对于数据本身，chunk server更加了解自身存储的情况，因此chunk的信息，chunk server说了算。这样避免了master中持久化一份数据分布的信息而带来的与实际情况不一致的情况。

master机器只有一台，为防止单点故障，肯定要准备备用机。一般会存在2台或者以上的shadow master。当master在本地写operation log的时候，也会让shadow master写一份。只有在全部都写成功的时候，操作才会继续。如果master的硬盘挂了，那么任意一台shadow master都能够挺身而出充当新的master。

关于master的容量大小。首先，一个chunk被设置成了64M，大大减少了一个文件所占用的chunk数量，因此，master中存储的数量也会大大减少。

其次，由于系统存储的是大文件，因此每一个文件，除了最后一个chunk以外，其他的chunk基本上都是存满的。一般64个bytes就可以存一个64M的chunk信息，master 容量不会是问题。

当operation log达到一定大小时，系统会切换到一个新的日志。老的日志则会被回放，并存储在硬盘上，这个操作被称作checkpoint。当master重启时，直接加载checkpoint生成的内容，并回放新的日志。

租约：对于每个chunk都有多个副本。当客户端要修改数据的时候，master通过使用租约来保证多个副本之间的一致性。master从所有副本中选出一个chunk server发放租约，得到租约的副本，称为primary，其余的就是secondary。租约时长60s。primary将同一个chunk的操作进行序列化。通过心跳包可以续租。master server可以主动收回租约。

1. **MapReduce**

MapReduce是一种编程模型，用于大规模数据集的并行运算。概念"Map（映射）"和"Reduce（归约）"，是它们的主要思想。它极大地方便了编程人员在不会分布式并行编程的情况下，将自己的程序运行在[分布式系统](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E7%B3%BB%E7%BB%9F/4905336)上。 当前的软件实现是指定一个Map（映射）函数，用来把一组键值对映射成一组新的键值对，指定并发的Reduce（归约）函数，用来保证所有映射的键值对中的每一个共享相同的键组。

关于MapReduce原理，我认为是利用分布式，进行分而治之的策略，将数据分割成64M（默认），然后又不同的节点去分析，这是每个节点独立运行（节点资源可以认为是独占，而单机多进程并发，还有争夺资源），这样化整为零，计算效率提高；最后将分析结果合并，这个结果要比最初的数据要小的非常多，不是一个数量级的。整体计算能力、资源使用率等等都要比单机要高。

* 关于master和worker的失效问题。

在一个很大的计算机集群中包含有成千上万台普通性能甚至次之的计算机，这么多的计算机出现宕机是很常见，所以需要考虑如何处理这种情况。对于挂掉的worker可以把它没做完的工作分给其他机器。那如果是master挂掉了呢？然后再选一个master，这是我觉的特别有趣的，机器之间居然会根据综合评价来选举。

1. **Bigtable**

本质上说，Bigtable是一个键值（key-value）映射。按作者的说法，Bigtable是一个稀疏的，分布式的，持久化的，多维的排序映射。

Bigtable的键有三维，分别是行键（row key）、列键（column key）和时间戳（timestamp），行键和列键都是字节串，时间戳是64位整型；而值是一个字节串。可以用 (row:string, column:string, time:int64)→string 来表示一条键值对记录。

行键可以是任意字节串，通常有10-100字节。行的读写都是原子性的。Bigtable按照行键的字典序存储数据。Bigtable的表会根据行键自动划分为片（tablet），片是负载均衡的单元。最初表都只有一个片，但随着表不断增大，片会自动分裂，片的大小控制在100-200MB。

行是表的第一级索引，我们可以把该行的列、时间和值看成一个整体，简化为一维键值映射。

列是第二级索引，每行拥有的列是不受限制的，可以随时增加减少。为了方便管理，列被分为多个列族（column family，是访问控制的单元），一个列簇里的列一般存储相同类型的数据。一行的列簇很少变化，但是列族里的列可以随意添加删除。列键按照family:qualifier格式命名的。这次我们将列拿出来，将时间和值看成一个整体，简化为二维键值映射。

时间戳是第三级索引。Bigtable允许保存数据的多个版本，版本区分的依据就是时间戳。时间戳可以由Bigtable赋值，代表数据进入Bigtable的准确时间，也可以由客户端赋值。数据的不同版本按照时间戳降序存储，因此先读到的是最新版本的数据。我们加入时间戳后，就得到了Bigtable的完整数据模型。

查询时，如果只给出行列，那么返回的是最新版本的数据；如果给出了行列时间戳，那么返回的是时间小于或等于时间戳的数据。

1. SSTable

BigTable数据在内部使用google SSTable文件格式存储。SSTable提供一个从键（key）到值（value）的持久化的、已排序、不可更改的映射（Map），这里的key和value 的都是任意的字节（Byte）串。对SSTable提供了如下操作：查询与一个指定key值相关的value，或者遍历指定key值范围内的所有键值对。从内部看，SSTable是一连串的数据块（通常每个块的大小是64KB，但是这个大小是可以配置的）。SSTable 会使用索引来加速查找，索引一般存储在数据的末尾位置。

1. Tablet

Tablet 包含了一个表中多行，一个tablet事实上是存储在一个机器上的一块信息。

其次，对于一个table来说，可能会分为多个tablet，而tablet又是由不同的sstable组成的，多个tablet可以共享一个sstable。

对于tablet 的查找，使用的是类似于文件系统的多级的索引结构：

使用一个三层的、类似于Ｂ+树的结构存储tablet的位置信息。

1. 基本的服务

Bigtable集群包括三个主要部分：一个供客户端使用的库，一个主服务器（master server），许多片服务器（tablet server）。

每个tablet一次分配给一个tablet服务器。master服务器记录活跃的tablet服务器、当前tablet到tablet服务器的分配、包括哪些tablet还没有被分配。一个tablet 服务器管理着多个tablet。

1. 读写操作

当片服务器收到一个写请求，片服务器首先检查请求是否合法。如果合法，先将写请求提交到日志去，然后将数据写入内存中的memtable。memtable相当于SSTable的缓存，当memtable成长到一定规模会被冻结，Bigtable随之创建一个新的memtable，并且将冻结的memtable转换为SSTable格式写入GFS，这个操作称为minor compaction。

当片服务器收到一个读请求，同样要检查请求是否合法。如果合法，这个读操作会查看所有SSTable文件和memtable的合并视图，因为SSTable和memtable本身都是已排序的，所以合并相当快。