**MapReduce**

MapReduce 是一个在多台机器上并行计算大规模数据的软件架构。主要通过两个操作来实现：Map 和 Reduce。

实际上，在一个 MapReduce 集群中，Master 会记录每一个 Map 和 Reduce 任务的当前完成状态，以及所分配的 Worker。除此之外，Master 还负责将 Mapper 产生的中间结果文件的位置和大小转发给 Reducer。

由于 Google MapReduce 很大程度上利用了由 Google File System 提供的分布式原子文件读写操作，所以 MapReduce 集群的容错机制实现相比之下便简洁很多，也主要集中在任务意外中断的恢复上。

在 MapReduce 集群中，Master 会周期地向每一个 Worker 发送 Ping 信号。如果某个 Worker 在一段时间内没有响应，Master 就会认为这个 Worker 已经不可用。

任何分配给该 Worker 的 Map 任务，无论是正在运行还是已经完成，都需要由 Master 重新分配给其他 Worker，因为该 Worker 不可用也意味着存储在该 Worker 本地磁盘上的中间结果也不可用了。Master 也会将这次重试通知给所有 Reducer，没能从原本的 Mapper 上完整获取中间结果的 Reducer 便会开始从新的 Mapper 上获取数据。

如果有 Reduce 任务分配给该 Worker，Master 则会选取其中尚未完成的 Reduce 任务分配给其他 Worker。鉴于 Google MapReduce 的结果是存储在 Google File System 上的，已完成的 Reduce 任务的结果的可用性由 Google File System 提供，因此 MapReduce Master 只需要处理未完成的 Reduce 任务即可。

整个 MapReduce 集群中只会有一个 Master 结点，因此 Master 失效的情况并不多见。

Google MapReduce 采用 Google File System 来保存输入和结果数据，因此 Master 在分配 Map 任务时会从 Google File System 中读取各个 Block 的位置信息，并尽量将对应的 Map 任务分配到持有该 Block 的 Replica 的机器上；如果无法将任务分配至该机器，Master 也会利用 Google File System 提供的机架拓扑信息将任务分配到较近的机器上。在某些情形下，用户所定义的 Map 任务可能会产生大量重复的中间结果键，同时用户所定义的 Reduce 函数本身也是满足交换律和结合律的。在这种情况下，Google MapReduce 系统允许用户声明在 Mapper 上执行的 Combiner 函数：Mapper 会使用由自己输出的中间结果 Partition 调用 Combiner 函数以对中间结果进行局部合并，减少 Mapper 和 Reducer 间需要传输的数据量。

**GFS**

Google设计并实现了GFS文件系统，一个面向大规模数据密集型应用的、可伸缩的分布式文件系统。GFS虽然运行在廉价的普遍硬件设备上，但是它依然了提供灾难冗余的能力，为大量客户机提供了高性能的服务。

在GFS系统架构中，GFS由Master-Slave架构构成。其中Master用来接收客户端的请求， Master节点管理所有的文件系统元数据。这些元数据包括名字空间、访问控制信息、文件和Chunk的映射信息、以及当前Chunk的位置信息。GFS存储的文件都被分割成固定大小的Chunk。在Chunk创建的时候，Master服务器会给每个Chunk分配一个不变的、全球唯一的64位的Chunk标识。Chunk服务器把Chunk以Linux文件的形式保存在本地硬盘上，并且根据指定的Chunk标识和字节范围来读写块数据。出于可靠性的考虑，每个块都会复制到多个Chunk块。

客户端将需要访问的数据文件名字以及需要访问文件的偏移量在本地进行计算，通过Chunk的大小，客户端计算得到需要访问的Chunk index。然后客户端将文件名字和需要访问的Chunk index发送给Master服务器。Master服务器接收这些请求，然后根据Master服务器上的Matadata信息、以及Chunk的映射信息。Master服务器将需要访问的Chunk位置(Chunk在哪些ChunkServer服务器上)和Chunk Handle发送给GFS client。GFS client根据每个Chunk的位置映射信息以及Chunk的地址请求偏移量访问相应的ChunkServer服务器，然后和ChunkServer服务器建立连接，进行数据传输。如果相应的ChunkServer服务器宕机或者故障，GFS Clinet可以访问相应的Chnnk的副本所在的ChunkServer服务器。GFS通过Master保存的元数据信息，降低了Master的负载，为大量客户机提供了高性能的服务。

GFS选择较大的Chunk尺寸有几个重要的优点。首先，它减少了客户端和Master节点通讯的需求，因为只需要一次和Mater节点的通信就可以获取Chunk的位置信息，之后就可以对同一个Chunk进行多次的读写操作。这种方式对降低我们的工作负载来说效果显著，因为我们的应用程序通常是连续读写大文件。即使是小规模的随机读取，采用较大的Chunk尺寸也带来明显的好处，客户端可以轻松的缓存数TB的工作数据集所有的Chunk位置信息。其次，采用较大的Chunk尺寸，客户端能够对一个块进行多次操作，这样就可以通过与Chunk服务器保持较长时间的TCP连接来减少网络负载。第三，选用较大的Chunk尺寸减少了Master节点需要保存的元数据的数量。这就允许我们把元数据全部放在内存中。

GFS分Chunk来进行存储主要有以下几个原因：

首先分Chunk来进行存储，主要原因是GFS被设计是用来存储大文件的。将一个大文件划分为多个Chunk小文件，并且这些多个Chunk是被分布在多个Slaves中。这样可以大大提高系统的整体吞吐量，可以让多个client并行的读取文件，多个client直接互不干扰。

GFS分Chunk来进行存储，还有一个更重要的原因就是。Chunk是可以进行副本的，通过大文件被划分为多个Chunk，每个Chunk还能有多个副本。当其中一个机器故障，机器中的Chunk被丢失。但是由于Chunk默认是有三个副本的，其中一个Chunk丢失，Client可以访问其他副本的Chunk。GFS通过副本的机制提供灾难冗余的能力，为分布式文件系统提供高性能、高可靠的服务。

**BigTable**

Bigtable是一个分布式的结构化数据存储系统，它被设计用来处理海量数据：通常是分布在数千台普通服务器上的PB级的数据。而且由于业务的需求，谷歌对于BigTable的要求是非常苛刻的，高吞吐量，低响应时间，且各个集群的配置大小也不尽相同。

BigTable的数据模型是非常简单的，虽然paper中定义中为结构化数据，但是这种简单的模型并不像关系型数据库的表一样有各种各样的约束，它只是一个多维度的有序Map，可以使用之所以说是多维度是因为在每一个Value中我们都可以去定义列，使得这个Map看起来就像是表一样。BigTable并不是一个独立的实现，它是基于谷歌一些其他的组件的，比如Chubby和GFS，因为BigTable为了能够支持PB级的数据，它必须要把数据文件存放于GFS中，而且Chubby优秀的存储元数据的能力也可以使得BigTable的实现更为简单。

但是我更关心的是缓存一致性的问题，Chubby使用一种通知客户端缓存失效，从而使得更新缓存的开销分布到多次操作来提高效率。因为Tables可能会分裂或者合并，那么BigTable是如何做到缓存一致的呢？答案其实就是在查看到所缓存的信息无法找到目标值的时候就去重新查找Chubby，因为Tables的改变写入Chubby才算是成功，所以Chubby总能看到最新的位置信息，如果客户端缓存是空的，那么寻址算法需要通过三次网络来回通信寻址，这其中包括了一次Chubby读操作；如果客户端缓存的地址信息过期了，那么寻址算法可能需要最多六次网络来回通信才能更新数据，因为可能Tables在元数据表中的位置都已经改变了。

还有一点很重要，就是Tablet并不是数据的最小单位，而是SSTable（Google SSTable），SSTable是一个持久化的、排序的、不可更改的Map结构，而Map是一个key-value映射的数据结构，key和value的值都是任意的Byte串。一个Tablet的全局视图由多个SSTable和一个memtable（最近提交的那些存放在一个排序的缓存中，我们称这个缓存为memtable）构成。

布隆过滤器：可以使用Bloom过滤器查询一个SSTable是否包含了特定行和列的数据，对于某些特定应用程序，我们只付出了少量的、用于存储Bloom过滤器的内存的代价，就换来了读操作显著减少的磁盘访问的次数。这里的具体做法paper中并没有详细的解释，我个人认为就是把局部性群组SSTable的列名哈希以后放入布隆过滤器（这样也可以看出这个列名是不可删除的）中，这样不需要读取磁盘就可以以极小的内存代价和O（1）的时间复杂度知道所需的列是否存在于当前SSTable中（我们可以忍受偶尔出现的误判，无非是多读一次磁盘）。

可以看出BigTable的实现至少在论文中看并不是多么的难以理解（当然从paper中也可以看出很多细节并没有展示出来，比如Tablet的数据冗余，如何进行根据key的分片这些重要的东西并没有提起），但是其设计确是巧妙绝伦，这也催生了令人惊叹的高效率于高吞吐量。同时它与GFS，MapReduce并称为谷歌的三驾马车，成为云计算的基石，我觉得不仅仅是云计算，这也是云存储的鼻祖。诚然这已经是十几年前的事情了，但是如今看来却仍让我们这些后辈惊叹不已。当然BigTable是闭源的，但是作为Hadoop的子项目的Hbase却是一个BigTable的开源实现，我还没有玩过Hbase，也并不了解，以后再说啦