**读Google File System后记**

——18301161 李垚

Google的伟大之处，不仅在于它创造了一个强大的搜索引擎，还在于它创造了三项革命性的技术——Google File System, Map Reduce, Big Table, 又被称为Google的三驾马车。这三篇依次发出的论文，给了IT行业关于分布式系统莫大的启发，相关的开源技术极大地普及了云计算之中非常核心的分布式技术。我在短短几周时间里，按照发布的时间顺序阅读了这三篇论文，虽然颇多不明白之处，但仍然收益颇丰。Google作为一家IT企业，其发表的论文与各高校研究所发表的论文最大的区别就是其实用性，不追求多么超前的技术突破，也没有多么翔实的数据分析，只是把实践产出的一套系统构建方法铺开了给你看，就能启发无数后来者，引领一个领域的进步。

Google File System是这三驾马车中的第一驾，它是一个分布式的文件系统。究其源头，无非是Google在实践中发现经常会产生多而大的数据，而这些数据又往往不适合分割成小数据来存储，于是采用分布式系统的方式，将大数据存储在大量的廉价硬件之上。显然这是一种以数量为代价的文件系统，以一种容错性高的方式来存储一些错误代价极大的大数据。

在这个文件系统当中，主要有以下几个着重解决的问题：错误繁多、大文件为主、追加式写文件为主、文件追加需高效、吞吐量需足够高。GFS采用了以下的几种方式来解决这些问题。其一，错误处理。从整个GFS以大量的廉价硬件来存储就可以看出，这个系统必然会发生大量的错误，硬件上的、软件上的、传输过程中的、甚至是人为的。Google并没有把这些错误当作异常，反而是把他们当作常态，从一开始就做了最坏的打算，因此GFS当中集成了持续监控、错误侦测、灾难冗余以及自动恢复的解决方案，以保证绝大多数错误都是系统可以承受的。首先Master服务器会周期性的心跳信息监控 Chunk服务器的状态，以保证每次执行操作时的Chunk服务器是可用且无误的；其次，如果发生Chunk服务器不可用的情况，系统会通过Checksum的方式来确认数据是否损坏；然后，如果数据损坏，系统会尽快地找到一个Chunk副本，并利用其中的数据来进行数据恢复；只有当所有的副本全部在数据恢复之前损坏，这个数据才会真正丢失，除此之外的错误都是可恢复的，这给了整个系统极大的容错性。

其二，存储结构。这个文件系统着重在解决存储大数据的问题，于是如何分布数据的存储就显得尤为重要。在GFS当中，Master服务器存储着整体文件结构信息、每个文件包含的Chunk信息、每个Chunk的存储位置（即存储在哪个或哪些Chunk服务器当中）、日志信息。其中Master服务器并不会持久性地存储所有Chunk的存储位置，而是在每次启动时，向所有的Chunk服务器询问其中存储了什么Chunk，而这也为持续监测以及时修复错误提供了便利。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的错误修复显得尤为重要。在GFS当中，Master会在本地保存操作日志，并且同时会在Shadow Master上存储这些日志，这预防了Master服务器可能出现的大多数错误。如果一台Master服务器不可用了，还可以及时将Shadow Master调出，替换掉原来的Master服务器。每个Chunk有64M的大小，这使得大文件需要占用的Chunk服务器个数大大减少，因此，Master服务器当中存储的信息数量也会大大减少。一般来说Master服务器只需64Byte即可存储一个Chunk的64M大小的基础信息，因此这样的结构能使Master的内存压力减小。当日志信息达到一定大小时，系统将会切换到一个新的日志，并把老的日志存储到硬盘里。这种切换日志的操作被称为checkpoint，当Master服务器重启时，将会直接加载checkpoint生成的内容，并回放新的日志。

其三，系统交互。要采用多个副本Chunk来存储同样的数据，就必然需要随时保证多个副本的数据一致性。在GFS当中，采用租约的方式来保证这一点。在要执行写入操作或者记录追加操作时，Master服务器会首先在所有符合要求的Chunk副本当中选出一个作为主Chunk，并发放一个60秒的租约；客户端此时便可以与主Chunk进行通信并进行变更操作；在租约超时前，对该Chunk的变更操作都由主Chunk进行序列化控制；在租约超时前，如果主Chunk预计在租约超时前变更操作不能完成，主Chunk还可以通过在与Master的心跳通信中申请续租；与此同时，Master服务器也可以主动收回租约来中止变更操作。于是在这样的交互体系当中，客户端只需要向Master服务器发起请求，然后Master服务器就会把操作权限交给Chunk服务器，并由其来与客户端进行下面的通信，与此同时，主Chunk会负责调度所有的从Chunk来达成一致，而从Chunk也会发出响应来表示一致性操作的成功与否。如果在此期间出现错误，则会有两种情况：一是Master服务器接收到了错误信息，这表示至少主Chunk上的操作是成功的，这样他才能发送回一个错误信息，但这也可以认为这一整个操作是不成功的，因为数据产生了不一致，于是客户端将会尝试重新进行一致性操作；二是Master服务器到租约到期也未能接收到任何信息，这说明在主Chunk上就已经出现了错误，那么Master服务器会在心跳确认数据完整性后，重新尝试设定主Chunk并执行接下来的操作。在这个过程当中，控制与数据操作被分化到Master和Chunk当中，客户端先向 Chunk Server 提交数据，再将写请求发往主Chunk，这么做 GFS 能够更好地利用网络带宽资源。

在这个交互体系下，如果要执行数据追加的操作就变得十分方便。如果主Chunk发现要追加的数据块会超出他的上限，则会首先追加部分数据以填满自身容量，并指挥从Chunk也如此执行，随后向Master服务器汇报，接着Master服务器就会考虑将剩下的数据添加到一个新的Chunk当中；如果主Chunk能完整地追加此数据块而不超出容量上限，则会直接追加，并指挥从Chunk执行同样的追加操作，最后返回给Master服务器成功的信息。而对于删除文件的操作，GFS的处理方式是在文件和Chunk两个层面上lazy地删除，也就是先将其重命名为另一个隐藏文件名，并记录其“被删除”的时间，过了一段时间后，如果在此期间用户没有要求还原此文件，Master服务器才会在周期性扫描的过程中将这些久远的被删文件彻底删去。这样的慢删除机制，既能避免一些类似在某些Chunk服务器上成功了，在其他 Chunk服务器上失败等问题，又让Master服务器能选择在空闲时来执行删除的操作，优化资源分配，还提供了删除操作的回退能力。

GFS当中还有很多细节值得思考，它为之后的Map Reduce以及Big Table奠定了基础，也催生了以HDFS为首的分布式系统实现，后者又为Hadoop奠定了基础，环环相扣地催生了大数据的广泛应用，可谓是大数据时代的开山大作之一。而通过查阅一些资料，我也发现在这十几年间，Google已经对这套系统做了许多的优化与改动，目前所公开的Colossus就是GFS的一种升级。在小学期使用过Hadoop，本学期学习过HBase过后，再回过头来看当时的这几篇论文，仍然为它们的严谨和体系性而感叹。