**谷歌GFS分布式文件系统读后感**

**18301137 王磊**

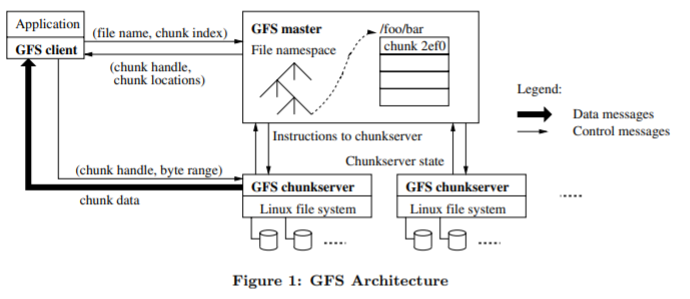
最近，我阅读了Google的GFS（Google File System）论文，本以为在之前了解过大数据的相关知识后，读本文时会相对比较轻松，但是读时才发现，里面的内容并不是很好理解，有些专用名词我也是第一次见，因此我也是边读论文边查资料，最后终于能把整篇论文大致地梳理清楚，并由衷的感叹GFS的强大与神奇之处，以下是我对GFS的一些总结与我个人的一些感想：

首先我概括一下读完文章后对GFS的理解。GFS是一个可扩展的分布式文件系统，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，通过软件的方式自动容错，它将服务器故障视为正常现象（而这也是区别于其他类似产品的不同之处），通过软件的方式自动容错，在保证系统可靠性和可用性的同时，大大降低系统的成本。另一方面，它拥有着数据完整性，用于大型的、分布式的、对大量数据进行访问的应用。它又有着有效的诊断工具，广泛而细致的诊断日志以微小的代价换取了在问题隔离、诊断、性能分析方面起到了重大的作用。GFS服务器用日志来记录显著的事件，如服务器停机和启动和远程的应答。远程日志记录机器之间的请求和应答，通过收集不同机器上的日志记录，对它们进行分析恢复，就可以完整地重现活动的场景，并用此来进行错误分析。这些都给用户提供了总体性能较高的服务，比较方便快捷。

另外，冗余也是GFS分布式文件系统的一个应用，冗余的使用可以使当有一个机器宕机时也不会停止该机器的服务，因为同一个文件至少有三个副本，当有机器宕机后，便会从其他的机器那里获取数据。GFS中文件被分割成很多块，使用冗余的方式储存于商用机器集群上，数据块冗余配置策略要达到最大的数据可靠性和可用性，最大的网络带宽利用率。因此，如果仅仅把数据的拷贝置于不同的机器上很难满足这两个要求，必须在不同的机架上进行数据备份。这样即使整个机架被毁或是掉线，也能确保数据的正常使用。这也使数据传输，尤其是读数据，可以充分利用带宽，访问到多个机架，而写操作，则不得不涉及到更多的机架。

论文中写到，GFS有几个主要的指导思想，它们分别是：1. 组件故障应当被视作日常而不是异常(基于大量廉价硬件)。 2. 文件的大小通常比较大(日志文件，几个G很常见) 3. 写文件的方式以追加为主，很少会覆盖(日志文件，一般都是追加文件) 4. 客户端的使用和文件系统是一起设计的，比较灵活。

以下是GFS的整体架构：



GFS由master和chunkserver组成，其中chunk（块）也是这篇文章的关键词，一个大文件被分成了很多个chunk，而一个chunk的大小为64MB，每个chunk会在多个chunkserver上保存作为冗余备份。Client端与master的交互流程大概是Client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，然后客户端再请求对应的chunk server操作数据。Master节点是单一的，master机器只有一台，在它的内存中存储了File namespace(文件目录结构)、每个文件包含的chunk以及每一个chunk存放在了那些chunkserver上。其中，对File namespace和每个文件包含的chunk的修改操作时，会先写操作日志，写完后才会进行操作内存，这样可以在机器宕机后，根据操作日志恢复到宕机前的状态。对于每个chunk存在了哪些chunkserver上来说，master不会持久化存储这些信息，而是在master启动时询问所有的chunkserver存放了哪些chunk，新的chunkserver加入时也会主动告诉master。这样的数据存储形式有很多好处，比如它解耦了元数据与数据，元数据只有master知道，因此对于master来说，为了保护好这些元数据，就必须要写操作日志。对于chunk数据本身，chunkserver说了算，这样就避免了master中持久化一份数据分布的信息而带来的与实际不一致的情况。

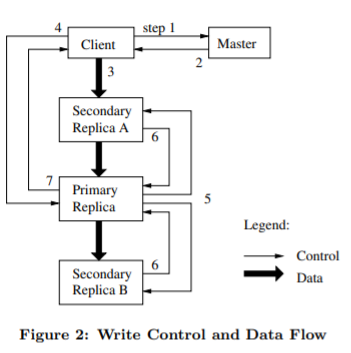
上面已经说到，master只有一台，要是master机器发生故障岂不就是GFS的灾难？而这些Google工程师早已经想到，为了防止单点故障，需要准备master备机，一般会有2台或者以上的shadow master。倘若master的硬盘挂掉，那么其他任意一台shadow master将会代替这个master继续工作。

接下来解释一下chunk为什么要设置成为64MB这么大的空间，这样做是因为存储的文件很多都是大文件，一个文件都能分成很多个chunk，并且除了最后一个chunk外，其他的所有chunk都是满的；另一个原因是chunk比较大时，那么所有文件的chunk数量就会变得很少，这样在master中的存储数量也会减少，因此这做减少了对master机器内存性能的需求。

GFS中还有一个名叫checkpoint的操作，相比于数据库中的checkpoint，数据库中的是checkpoint前的事务全都成功提交或者回滚，而GFS中的checkpoint是指当master操作日志达到一定大小时，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。这样当master重启时，直接加载checkpoint生成的内容，并回放新的日志即可。这样做减少了重启时日志回放所需要的时间。

接下来介绍客户端修改数据的流程（流程图如下），在介绍之前我们要先知道一个概念——租约。每个chunk都有很多副本，当Client端要修改数据时，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chunkserver发放租约，得到租约的副本被称为primary，其余的便是secondary，租约的时长为60秒，primary将同一个chunk的操作进行序列化。通过心跳包可以进行续租，master也可以主动收回租约。具体Client端修改数据流程如下:

1. Client端请求server某个chunk。
2. master告诉Client拥有这个chunk的primary和secondary分别是哪些chunkserver。
3. Client将自己的操作数据传给其中某个chunkserver，chunkserver在收到数据之后，立即推给最近的另外一台没有接受过数据的chunkserver，直到所有chunkserver都有这个数据。
4. Client告诉primary进行操作。
5. primary告诉其他secondary进行操作。
6. 若secondary执行成功，则整个操作成功。
7. 若操作失败，由primary告诉Client，并进行重试。



最后是一个追加记录的功能，也是最常使用的一种写入方式。记录追加操作是原子操作，多个客户端可以同时向primary请求执行这个命令，这些命令会被放在队列当中，由primary一个个来完成追加的操作（多生产者/单消费者模型）。如果primary发现追加的内容，若追加内容大于该chunk剩余的空间，就会将chunk剩余的部分填充满，并通知secondary也进行同样的操作，并通知Client端需要使用新的chunk来操作。Client端之后重新发起请求。当chunk的大小够用时，则写入chunk，并通知secondary同样写入，并将结果返回给Client端。若primary执行失败，则直接返回失败并重试；若primary执行成功，但secondary中有一或多个返回失败，则返回Client端失败并让其重试，但是primary和成功的secondary不回滚之前的操作。因此，当客户端重试的时候，primary中会产生两条相同的记录。几个chunk server之间可能会发生不一致的情况。这种不一致的情况允许存在，也就是说GFS放宽了server端一致性的要求。Client端需要对这种情况有所准备，需要有能够过滤重复数据的能力。

在读完Google的GFS分布式文件系统后，我为其强大的功能与极佳的性能感到震撼，从某种程度上说，Google的三驾马车奠定了风靡全球的大数据算法的基础，为大数据时代开创了一条道路，也为大数据时代的发展做出了巨大贡献。虽然看了这么久的论文，也查了很多资料，但还是对于某些概念不太理解，今后我也会将MapReduce与BigTable两篇论文进行阅读，希望在不久的将来自己可以跟上大数据时代的潮流，学习到大数据的前沿技术。