**Google三篇论文读后感**

蒋泽群-18301130

通过阅读和学习Google关于GFS、MapReduce和BigTable的三篇技术论文，我对大数据基本结构原理和架构有了一定的了解，接下来我就简单讲一下我读完这三篇论文的收获以及读完之后的感受。

1. GFS

GFS(Google File System)是Google开发的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，用来解决处理大数据的存储问题。它实现的需求主要如下：

* 基于廉价的组件，组件失效为常态事件而非意外事件，这就要求GFS能实现持续的监控、错误侦测、灾难冗余、自动恢复等机制来实现容错功能。
* 系统需要操作的文件巨大。
* 系统支持两种读操作：大规模的流式读取和小规模的随机读取。
* 系统对绝大部分文件的修改是采用在文件尾部追加数据，而非覆盖原有数据。
* 应用程序和文件系统 API 的协同设计提高了整个系统的灵活性。 引入原子性的记录追加操作，不需要额外同步操作保证数据一致性。

为了存储超大型文件，GFS采用了一个单独的 GFS Master 节点和多台GFS Chunk服务器相结合的主从结构的架构。Master 节点是保存所有文件系统元数据的地方，比如文件名、文件与Chunk的映射信息、当前Chunk的位置信息等等，相当于索引信息。Master同样管理系统范围内的活动并发送心跳信息与每个Chunk服务器通讯；GFS Chunk 服务器是用于存储大文件分割形成的固定大小的Chunk，其底层是由Linux文件系统实现的。Chunk（默认为64MB）可以看作一个大容量的Block（默认为1024Byte），GFS使用Chunk的原因是一个大文件可能需要使用数以万计的Block来保存，这也就意味着同时需要上万条索引信息显示每个Block的位置，而如果这里使用一个Chunk来替代的话，只需要存储一个索引就够了。Chunk的使用节省了文件原始信息的大小和空间，同时也节省了传递信息的流量。除此之外，为了减少Master和Chunk Server之间的通信，GFS选择在Chunk Server存储本地数据的偏移量而不是直接存储在Master中，这避免了每次Chunk Server的数据变化都需要通知Master的问题。GFS在保存数据中也很难避免组件失效的情况，为了减少这种情况所带来的损失，GFS对Chunk的保存就需要存在多个Chunk Server中，也就是建立副本。建立副本也可以用于实现数据损坏或丢失后的修复机制，例如当Chunk Server检测到Chunk04错误后，可以向Master求助寻找也存有Chunk04的Chunk Server实现错误数据的恢复。同样的当一个Chunk Server宕机导致其中的Chunk全部丢失，这时的恢复要以存活副本数少的Chunk优先以保证修复的是最紧急的Chunk。

作为一个文件系统，除了实现文件的保存，读和写的操作更为重要。GFS文件的读操作首先需要GFS client向Master发送请求，其内容是要读的文件名以及它的索引块，Master根据请求的信息找到该索引块所在的服务器位置并赋予Client访问权限，之后Client从最近的一个服务器获取该块的数据。 GFS文件的写操作，因为文件拥有副本，所以要同时在多个Chunk Server上写数据。当GFS client向Master发送写文件的请求，Master会告诉client哪些Chunk Server存放了副本并指定拥有副本的其中一个Chunk Server作为主结构，主结构主要用于向其他Chunk Server发送写的指令，并在所有Chunk Server完成后向client回复。值得注意的是，当一次写操作发生了错误，GFS没有修复的机制，而是直接进行向client请求重试操作。

1. MapReduce

MapReduce是针对分布式并行计算的一套编程模型，MapReduce的实现基于GFS存储数据。MapReduce是由Map（映射）和Reduce（归纳）组成，其编辑模型为用自定义的Map函数接收一个输入的key/value pair值，然后产生一个中间key/value pair值的集合。MapReduce库把所有具有相同中间key值的中间value值集合在一起后传递个reduce函数。其操作流程如下：

* 用户程序将输入数据分为M个数据片段。
* 用户程序在集群中创建一个Master程序和若干个Worker程序。其中Master程序负责将M个Map任务和R个Reduce任务分配给Worker程序，一个Worker只负责一个Map任务或者一个Reduce任务。
* 被分配了Map任务的Worker程序读取相关的输入数据片段，解析出key/value pair。之后将key/value pair数据传递给用户自定义的Map函数。Map函数生成并输出中间key/value pair，缓存在内存中。
* 缓存在内存中的中间key/value pair通过分区函数分为R个区域，周期性写入本地磁盘。
* 之后将写入本地磁盘的存储位置传回给Master， Master再将这些存储位置传输弄个Reduce Worker。
* Reduce Worker程序收到中间key/value pair数据的存储位置之后，使用RPC从Map Worker所在的磁盘读取这些缓存数据。
* Reduce Worker读取所有的中间key/value pair数据，根据key值进行排序使得相同key值得数据聚合在一起。
* Reduce Worker将key值对应的value值集合传递给用户自定义的Reduce函数。Reduce函数输出被追加到所属分区的输出文件。
* 当所有的Map和Reduce程序完成之后，Master唤醒用户程序。

1. BigTable

BigTable是一个以GFS为底层的分布式数据存储系统。BigTable是非关系型数据库，本质上是一个稀疏的、分布式的、持久化存储的多维度排序Map，它采用行键、列键和时间戳进行索引，其中行键为一级索引、列键为二级索引、时间戳为三级索引。BigTable也提供增删改等操的API，并采用了原子更新操作，可以将多个相关联的动作合起来一起操作。BigTable的底层数据文件和日志文件是使用GFS来存储的，其文件格式为Google SSTable文件格式，其大小与硬盘中的块相匹配。为了加快写数据的速度，SSTable首先会存储在内存中，这样做的好处是由于内存存储是随机存储，相较于硬盘更容易实现根据索引来不断改变key的排序。当内存表过大，也就是到达硬盘中一个块的大小时会从内存写入硬盘。

BigTable的架构由三部分组成：库函数、一个主服务器和多个Table服务器。库函数负责服务器和客户端的通信，提供各种操作的控制。客户端不会与主服务器交互，而是直接和Table服务器通信进行读写的操作，主服务器的作用是处理原始信息和处理负载均衡等任务。Tablet的位置通过维护元数据表实现，每个元数据表分别记录了对应的表的位置。元数据表存放在分布式锁服务Chubby中，保证了数据的一致性。

1. 感想

粗略的读完Google的三篇论文，其算法的实现是具有启发性的，同时也让我对分布式系统和大数据算法有了一定的了解。对于大数据的处理，Google采用的想法就是采用多层结构的设计将底层的每一的部分设定为一个固定的、合适的大小，包括GFS对文件的处理以及BigTable对表的处理，这其实让我联想到学习《操作系统》是对文件存储的处理，其思路其实是相似的。同时，对于分布式系统我也有了更多的认识。比如，对于一个主从结构，主服务器要不断呼叫其他服务器以判断其他服务器的正常运行。在分布式系统里，任务处理失败，系统会选择不去处理，也没有应对错误处理的机制，而是直接重新执行任务，这样实现不但简单而且能避免更多的错误发生，这样的想法也在论文中多次出现。受制于自身的专业知识储配，这三篇论文我还不能做到完全理解，但我从中了解大数据的相关知识，处理这些问题使用的方法，拓宽了视野，丰富了自己的知识积累，相信对我未来遇到的问题会有所帮助。