BigTable 读后感

18301088 张正阳

提到 Google，就不得不谈谈它的分布式系统。作为当今世界上最发达的搜索引擎，它创造了 3 项革命性技术：GFS、MapReduce 和 BigTable，即所谓的 Google 三驾马车。

其中，GFS 是一个可扩展的大型数据密集型应用的分布式文件系统，该文件系统可在廉价的硬件上运行，并具有可靠的容错能力，该文件系统可为用户提供极高的计算性能，而同时具备最小的硬件投资和运营成本。而基于 MapReduce 编写的程序是在成千上万的普通 PC 机上被并行分布式自动执行的。它将所有服务器中的处理器有效地利用起来计算保存在谷歌文件系统中的海量数据并得到想要的结果。

BigTable 则是建立在 GFS ，Scheduler ，Lock Service 和 MapReduce 之上的。就像文件系统需要数据库来存储结构化数据一样，GFS 也需要 BigTable 来存储结构化数据。 每个 Table 都是一个多维的稀疏图。为了管理巨大的 Table，把 Table 根据行分割，这些分割后的数据统称为：Tablets。每个 Tablets 大概有 100-200 MB，每个机器存储 100 个左右的 Tablets。每个 Tablet 服务器都管理一个 Tablet 的集合（通常每个服务器有大约数十个至上千个 Tablet）。每个 Tablet 服务器负责处理它所加载的 Tablet 的读写操作，以及在 Tablets 过大时，对其进行分割。”，从这些可以大致了解到 BigTable 的工作原理。底层的架构是：GFS。由于 GFS 是一种分布式的文件系统，采用 Tablets 的机制后，可以获得很好的负载均衡。比如：可以把经常响应的表移动到其他空闲机器上，然后快速重建。以下是读过 BigTable 论文后学到的一些东西：

BigTable 是一个稀疏的，分布式的，持久化存储的多维度排序 Map，是谷歌设计的数据存储系统，它是全球化的、分布式的、持久化存储的、多维度排序的(数个层级)、可以被部署在几千台计算机上用来处理海量数据的一种非关系型的数据库。它还是一个分布式的结构化数据存储系统，在 Hadoop 中是以 HBase 来实现同样的功能。这两个存储系统其设计都是针对海量数据（通常是分布在数千台普通服务器的 PB 级的数据）处理的。HBase 是一个高可靠的列式数据存储系统。目前 Google 的很多项目都使用 BigTable 存储数据，包括 Web 索引，Google Earth，Google Finance。

1. 行

表中的行关键字可以是任意的字符（目前支持最大 64KB 的字符串）；

对同一个行关键字的读或者写操作都是原子的；

BigTable 通过行关键字的字典顺序来组织数据。表中的每一个行都可以动态分区。

每分区叫做一个 Tablet，Tablet 是数据分布和负载均衡调整的最小单位。

2. 列族

列关键字组成的集合叫做“列族”，列族是访问控制的基本单位；

列族在使用之前必须先创建，然后才能在列族中任何的列关键字下存放数据；

列族创建后，其中任何一个列关键字下都可以存放数据。一张表中的列族不能太多（最多几百个），且列族在运行期间很少改变。相反，一张表可以有无限多个列。

列族的名字必须是可打印的字符串。访问控制，磁盘和内存的使用统计都是在列族层面进行的。

3. 时间戳

在 BigTable 中，表的每一个数据项都可以包含同一份数据的不同版本；

不同版本的数据通过时间戳来索引；

BigTable 时间戳的类型是 64 位整型；

BigTable 可以给时间戳赋值，用来表示精确到毫秒的“实时”时间；

用户程序也可以给时间戳赋值；

数据项中，不同版本的数据按照时间戳倒序排序，即最新的数据排序在最前面。

除此之外，BigTable 还提供了建立和删除表以及列族的 API 函数。BigTable 还提供了修改集群，表和列族的元数据的 API，比如修改访问权限。客户程序可以对 BigTable 进行如下的操作：写入和删除 BigTable 中的值，从每个行中查找值，或者遍历表中的一个数据子集。

BigTable 还可以和 Mapreduce 程序一起使用，Mapreduce 是 Google 开发的大规模并行计算框架。通过开发了 Wrapper 类,通过使用这些 Wrapper 类,Bigtable 可以作为 MapReduce 框架的输入和输出。

BigTable 还依赖一个高可用的,序列化的分布式锁服务器组件,叫做 Chubby。BigTable 使用 Chubby 完成以下几个任务：

（1）确保在任何给定的时间内最多只有一个活动的 Master 副本；

（2）存储 BigTable 数据的自引导指令的位置；

（3）查找 Tablet 服务器，以及在 Tablet 服务器失效时进行善后；

（4）存储 BigTable 的模式信息（每张表的列族信息）；

（5）以及存储访问控制列表。

还记得老师上课讲到的 BigTable 和 HBase 的联系，我也试着去理解 HBase 和 BigTable 的联系与区别。首先，BigTable 如何使用 Chubby 跟 HBase 如何使用 ZooKeeper 有很多异曲同工之处。但有一个区别就是：HBase 并不把 Schema 信息存储在 ZooKeeper 中。它们都非常依赖锁服务的正常运作。我们经常低估当 ZooKeeper 无法取得足够的资源去作出实时回应时的后果。宁可让 ZooKeeper 集群运行在相对较老旧的但是什么别的事都不干的机器上，而不是运行在已被 Hadoop 或者 HBase 进程搞得不堪重负的机器上。一旦你的 ZooKeeper 没有足够的资源提供服务，就会引发多米诺骨式的效应，HBase 将会挂掉—包括 master 节点。

另外一个重要区别是：ZooKeeper 并不是一个像 Chubby 一样的锁服务系统，但是目前为止，这并不是 HBase 所关心的。ZooKeeer 提供一个分布式的协调服务，让 HBase 可以选举出 Master 节点。它也可以提供用以表示状态或者某个动作需要的信号量。当 Chubby 生成一个锁文件来表示一个 tablet 活动的，与此相对应的一个 Region server 会在 ZooKeeper 中生成一个节点来表示自己的存在。这个节点创建以后，只要 ZooKeeper 不挂，它会一直存在。在 BigTable 中，当一个 tablet server 的锁文件被删除时就表示与这个 tablet server 的租约失效。在 HBase 中，因为 ZooKeeper 相对少点限制的架构，这种行为会被处理得有所不同。它们只是语义上有所差别，并不意味着谁优谁劣，仅仅有所不同而已。

虽然 Donald Knuth 曾经说过“过早优化是万恶之源”，但在产品代码基本稳定的时候，做一定优化，还是非常有帮助的。BigTable 为了提高性能也做出了很多优化。比如可以通过将多个 Column Family 组合成一个局部性群组，而且系统对 Tablet 中的每个局部性群组都会生成一个单独的 SSTable。通过局部性群组这个机制，能将多个比较类似的 Column Family 整合到一起，这样做有两个好处：其一是能减少数据的读取；其二是提升处理速度。还有，可以以局部性群组为单位设定一些有用的调试参数。比如，可以把一个局部性群组设定为全部存储在内存中。Tablet 服务器依照惰性加载的策略将设定为放入内存的局部性群组的 SSTable 装载进内存。加载完成之后，访问属于该局部性群组的列族的时候就不必读取硬盘了。这个特性对于需要频繁访问的小块数据特别有用：在 BigTable 内部，利用这个特性提高 METADATA 表中具有位置相关性的列族的访问速度。

在关系型数据库的时代，由于压缩率底，而且在性能上提升幅度也偏低，使得压缩技术对关系型数据库而言，只能算是画龙点睛，但是对基于 Column 的数据库而言，由于其是将一个 Column 或者几个近似的 Column 的数据放在一起存放，所以在压缩率上面非常惊人，甚至到 1：9，特别是当今 CPU 的速度远胜于 I/O 的传输速度和存储容量的时代，通过增加少许用于解压缩的 CPU 时间来大幅降低读取和传输数据的时间，这对性能有非常明显的提升。

BigTable 也采用了压缩机制，比如，客户程序可以控制一个局部性群组的 SSTable 是否需要压缩；如果需要压缩，那么以什么格式来压缩。每个 SSTable 的块都使用用户指定的压缩格式来压缩。假如只需读取一个 SSTable 中的部分数据，那么就可只需那些部分数据进行压缩，而不必解压整个文件。BigTable 采用了两回合可定制的压缩方式。第一遍采用 Bentley and Mcllroy 方式，这种方式在一个很大的扫描窗口里对常见的长字符串进行压缩；第二遍是采用快速压缩算法，即在一个 16KB 的小扫描窗口中寻找重复数据。两个压缩的算法都很快，在 2006 年左右的 X86 硬件设备上，这套机制的压缩的速率达到 100-200MB/s，解压的速率达到 400-1000MB/s。

阅读过这篇论文，我心中感慨万千。整个 BigTable 设计符合大部分大数据程序的需求，打破了关系型数据库的结构化存储，能够部署在成千上万台服务器上，可以存储 PB 级数据，对整个互联网行业的快速发展提供了坚实的理论基础与成功案例。然而，我也只学到了 BigTable 的皮毛，真正想学到其精髓还得在日后的不断实践探索与思考中慢慢感悟。同时感谢老师给予我们这样一次特别的学习机会，没有让我们陷入在程序实现的苦恼中，而是领悟这样一篇处于技术前沿的论文，使我获益匪浅，未来的方向也逐渐清晰起来。