Google 三驾马车之 BigTable 读后感

18301034 陈佳悦

现今我们常听到的一个词“大数据”，其技术起源正是 Google 于 2004 年前后发表的三篇论文，也就是俗称的大数据 “三驾马车”。它们分别是分布式文件系统 GFS、大数据分布式计算框架 MapReduce 和 NoSQL 数据库系统 BigTable。搜索引擎主要就做两件事情，一个是网页抓取，一个是索引构建和排序，而在这个过程中，有大量的数据需要存储和计算。这 “三驾马车” 其实就是用来解决这个问题的，也就是，一个文件系统、一个计算框架、一个数据库系统。下面，我将就“三驾马车”之一 BigTable 来作一些总结并谈谈我的看法。

一、 Bigtable 概述

BigTable 是一个分布式存储系统，其结构为一个大表——一个大小为 PB，分布在几万台机器中的表。它被设计用于存储诸如：数十亿个 URL 之类的条目，每个页面具有多个版本、超过 100TB 的卫星图像数据、数以亿计的用户，以及每秒执行数千次查询。自 2005 年以来，BigTable 已经在许多 Google 服务中使用。Apache 项目在 Hadoop 核心之上创建了一个开源版本 HBase。因此李老师在课堂上提到的很多关于 HBase 的内容基本都和 BigTable 的特性一致。Apache Cassandra，最初是在 Facebook 上开发的，为搜索引擎提供动力，类似于 BigTable，它具有可调的一致性模型，没有主服务器（中央服务器）。

BigTabe 是用半结构化数据存储设计的。它是由行键、列键和时间戳索引的大地图。映射中的每个值都是由应用程序解释的字节数组。每次对行读或写数据都是原子性的，而不管在该行中读或写多少不同的列，所以很容易画出一张简单的表格。

二、 Bigtable 特性

BigTabe 有如下几个特性：

1. Map 映射

映射是一个关联数组，一种允许快速查找对应键值的数据结构。BigTable 是（key，value）键值对的集合。其中，键标识行，值是一组列。Bigtable 的键有三维，分别是行键（row key）、列键（column key）和时间戳（timestamp），行键和列键都是字节串，时间戳是 64 位整型；而值是一个字节串。可以用 (row:string, column:string, time:int64)→string 来表示一条键值对记录。

2. Persistent 持久性

由于数据存储在磁盘上，当程序结束或数据入口关闭后，保存在 map 中的数据会被持久化。这个和其他的持久化存储方式区别不太大。

3. Distributed 分布式

BigTable 构建在分布式文件系统上，也就是将数据分布在许多独立的机器中，以便底层文件存储可以在独立机器阵列之间传播。在谷歌中，BigTabe 是在 GFS（谷歌文件系统）之上构建的。Apache 开源版本的 BigTable——HBase，构建在 HDFS（Hadoop 分布式文件系统）或 Amazon S3 之上。该表在行之间划分，由服务器管理相邻行的组。行本身从不分布。数据以类似的方式在多个参与节点中复制，以便数据在基于独立冗余磁盘阵列的系统中的光盘之间进行条带化。

4. Sparse 稀疏

该表是稀疏的，这意味着表中不同的行可以使用不同的列。对于特定行，很多列是空的。一个给定的行在每个列族中都可以有任意数量的列，或者压根没有。稀疏的另一种类型是行间隙，这意味着 key 之间可能存在间隙。

5. Sorted 有序

与大多数 Map 实现不同，在 BigTable 中，键值对以严格的字母顺序保存，也就是说，键“aaa”的存储行应该在键“aab”的旁边，并且距离具有键“zzz”的存储行非常远。这有助于将相关数据紧密地放在一起，通常在同一台机器上——假设一个结构键以这种方式排序将数据聚集在一起。例如，如果将域名用作 BigTable 中的键，则将它们反向存储，以确保相关域紧密地结合在一起是有意义的。

因为这些系统往往是特别庞大的，并且数据是分布式存储，所以这种排序功能其实非常重要。具有类似键的行的空间优势确保了当一个操作必须扫描表格时，查询最感兴趣的内容将彼此靠近。需要注意的是，在 HBase 和 BigTable 中的 value 是不被排序的，只有 key 被排序。除了这个，其他的和普通的 map 实现是一样的。

6. Multidimensional 多维度

Bigtable 不是关系型数据库，却沿用了很多关系型数据库的术语，像 table(表)、row（行）、column（列）等。但若将其与关系型数据库的概念对应起来往往会误入歧途，难以理解论文。

此处所说的“表”是按行索引的，每行包含一个或多个命名列族。在创建表时就定义好列族。在一个列族中，可以有一个或多个命名列。列族中的所有数据通常是相同类型的。BigTable 的实现通常将列家族中的所有列压缩在一起。列族中的列可以即时创建。行、列族和列在识别数据方面提供了三级命名层次结构。例如：

```
table{
// ...
"rise" : { //一行
 "A" : { //列族 A
   "Vin" : {sth.}, //一列
   "Ivan" : {sth.}
 },
 "B" : { //列族 B
```

```
    "" : {sth.}
   }
  },
  // ...
}
```

7. Time-based 时间戳

时间是 BigTABLE 数据的另一个维度。Bigtable 允许每个列族保留数据的多个版本版本区分的依据就是时间戳。如果应用程序没有指定时间戳，它将检索列族的最新版本。或者，它可以指定一个时间戳，并获得早于或等于该时间戳的最新版本。例如：

```
 table{
 // ...
 "rise" : { //一行
  "A:Light" : { //一列
    16 : "x", //一个版本
    4 : "g"
  },
  "A:Bill" : { //一列
    15 : "z",
  },
  "B:" : { //一列
    7 : "w"
    5 : "y"
   }
 },
 // ...
}
```

查询时，如果只给出行列，那么返回的是最新版本的数据；如果给出了行列时间戳，那么返回的是时间小于或等于时间戳的数据。比如，我们查询"rise"/" A:Light"，返回的值是"x"；查询"rise"/"A:Bill"/6，返回的结果就是"g"；查询"rise"/" A:Bill"/2，返回的结果是空。

三、 Bigtable 工作原理

BigTable 包括三个主要的组件：链接到客户程序的 lib、一个 Master 服务器和多个 Tablet 服务器。

1. Master 服务器负责：

   （1）为 Tablet 服务器分配 Tablet

   （2）检测新加入或者过期失效的 Tablet 服务器

   （3）对 Tablet 服务器进行负载均衡

   （4）对保存在 GFS 上的文件进行垃圾收集

   （5）处理模式信息的修改（新建表、列族及访问控制信息）

2. Tablet 服务器

   Tablet 服务器负责管理一个 Tablet 集合（通常每个服务器有大约数十个至上千个 Tablet)。Tablet 服务器负责处理它所加载的 Tablet 的读写操作，以及分裂和合并。

3. Tablet 定位

   BigTable 使用类似 B+树的结构存储 Tablet 的位置信息。

   第一层是 Chubby 中的自引导信息，指向 Root Tablet。

   第二层是 Root Tablet，其中的每一个 Row 指向一个 MetaData Tablet。

   第三层是 MetaData Tablet，其中每一个 Row 指向一个存放数据的 Tablet，Row key 由该 Tablet 所在的表名和该 Tablet 最后一行组成。MetaData 的每一行数据大小大约是 1KB。

   其中 MetaData 和 Tablet 可以自动分裂，Root Table 永远不会被分裂，保证索引信息永远不超过三层。

   客户端会缓存 Tablet 的位置信息，当缓存为空时，需要三次网络通信。当缓存

过期时，最多需要六次网络通信。

4. Table 分配

任何时刻、一个 Tablet 只能分配给一个 Tablet 服务器。Master 服务器跟踪记录当前活跃的 Tablet 服务器，Tablet 的分配状态以及未分配的 Tablet。在需要时，进行 Tablet 的分配。

当启动一个 Master 服务器时，需要执行以下步骤获取系统当前状态信息，为 Tablet 分配作准备：

（1）从 Chubby 获取 Master 锁，阻止创建其他的 Master 实例；

（2）扫描 Chubby 服务器文件锁存储目录，获取当前正在运行的 Tablet 服务器列表；

（3）和所有正在运行的 Tablet 服务器通信，获取每个 Tablet 服务器上的 Tablet 分配信息；

（4）扫描 MetaData 表获取所有的 Tablet 集合，若发现未分配的 Tablet，则将其加入未分配 Tablet 集合，等待合适的机会分配。

5. Tablet 服务

Tablet 的持久化信息（redo log 和 SSTable）存储在 GFS 上。

BigTable 通过 Redo Log 进行故障恢复， 当执行一个更新操作时，首先会先写 Redo Log，成功后，再修改内存数据结构 memtable。当 memtable 大小达到某个阈值后，为了减少内存消耗，会进行 compaction 操作，将 memtable 转换成 SSTable，同时记录下 Redo Point。

当恢复一个 Table 时，Tablet 服务器首先从 MetaData 表中读取元数据（SSTable 文件列表，Redo Log 以及一系列 Redo Point）。Table 服务器把 SSTable

的块索引加载到内存，然后从 Redo Point 之后对 Redo Log 进行 replay，从而构建 memtable。

进行读写操作时会先进行权限验证，一个读操作需要读取所有的 SSTable 的 memtable 的结果进行合并。

6. Compaction 压紧

minor compaction：memtable -> SSTable

major compaction：memtable + SSTable -> SSTable

minor compaction 主要是两个目的：减少 Tablet 使用的内存，减少 Tablet 恢复时读取的 log 量。

major compaction 可以减少文件占用的空间，同时可以回收已经删除的数据占用的资源。

四、 HBase 与 Bigtable 的对比

HBase 是一个高可靠、高性能、面向列、可伸缩的分布式数据库，是 BigTable 的开源实现，主要用来存储非结构化和半结构化的松散数据。HBase 的目标是处理非常庞大的表，可以通过水平扩展的方式，利用廉价计算机集群处理由超过 10 亿行数据和数百万列元素组成的数据表。

HBase 与 Bigtable 的底层技术对应关系为如下：

Bigtable	HBase
GFS	HDFS
MapReduce	Hadoop MapReduce
Chubby	Zookeeper

除了底层技术，HBase 与 BigTable 的特性仍有一些细微的差别，如 BigTable 存储文件格式为 SSTable，HBase 为 HFile；BigTable 可以让存储文件直接映射到内存而 HBase 不支持；BigTable 支持单 Master ，而 HBase 支持多个 Master——”热”待命模式工作，多个 Master 都侦听 ZooKeeper 上的 Master 节点等。

五、 总结

以上便是论文中原理部分的大致介绍。通过拜读 Google 的“三驾马车”之 Bigtable，我对这个领域有了一点浅显的了解。虽然程度可能仅局限于摸到入门的边框，但接触到真正的论文，才发现这些比较前沿的理论也不是如此地“高高在上”，还是可以被我们理解和运用的。“万丈高楼平地起”，也许刚刚接触一个新事物会感到陌生和困难，但只有多阅读，多思考，多实践，才能更好地在技术的道路上走下去。