Big Table 论文阅读

Big table是一种稀疏，持久，分布式，多维度的有序Map

- 稀疏性 : 每一行的列组成可以不相同，不是严格的结构，而是一种稀疏的结构
- 持久性：依托于GFS，将数据持久化的存储在硬盘中的SSTable文件中
- 分布式：master server调度，chubby提供分布式锁服务，将tablets分给不同的tablet server实现分布式服务
- 多维度：通过行键，列键，时间戳三级索引多维度的实现存储
- 有序：通过行键的字典序进行排序，时间戳为倒排，即新的数据在上方

Big table的四个设计目标:

- 广泛的适用场景
- 横向扩展能力
- 高性能
- 高扩展性

Big table 数据模式&&核心概念:

Big table表中的数据通过一个行关键字，一个列关键字，一个时间戳进行三级索引

存储逻辑可以用这种形式表述:

`(row:string, column:string time:int64)→string`

- **行 Row：行关键字为第一级索引**
    - 行关键字可以是任意的字符串，但是大小不能超过64KB
    - 表中数据按照行关键字的字典序进行排序
    - 同一地址域的数据存放在表中连续的位置
    - 倒排便于数据压缩，可以大幅提高数据的压缩率
    - Big table支持行级事务级别，支持row级别操作的原子性
    - 同一行可以包含一个或多个列，不同行列的组成可以不同

  注意：对于一个网站例如`www.cnn.com`其存储在Big table的格式为`com.cnn.www`,这样倒排的好处是对于同一域名的内容，我们可以进行更加快速的索引

- **列关键字 Family：Qualifier： 列关键字为第二级索引**
    - **列族 Column Family :**
        - 族名必须有意义，标识符可以随意起
        - 是具有相同类型的列的集合，有两点优势:
            - 相同类型的列放在一起，可以提高压缩效率
            - 将相似的信息放在一个列族中的不同列，提高了读数据效率
        - 必须先创建列族，才能在该列族下创建任意列以及进行数据的读取

        - 列族的数量最好不要超过数百，且操作过程中尽量少改变，但可以无限扩展列的数量
        - **列族是访问权限控制的最小单元**
- **时间戳 Timestamp ：时间戳为第三级索引**
    - Bigtable中的时间戳是64位整型数，具体的赋值方式可以用户自行定义
    - 每一个Column Key可能关联多次更新，因此，Bigtable使用Timestamp来标识不同的版本
    - 同一个Column Key的多个版本按Timestamp倒序存放，这样查询时总是先读取到最新的版本
    - 每一个Column Family允许用户配置最多保留的版本数量，超出的版本将会被清理掉
- **表块 tablet ： Table的横向数据分区（分片）称之为Tablet， 提供了可伸缩性，为分布式提供了基础**
    - Tablet是一个连续的Row Key区间
    - Tablet是数据分布与负载均衡的基本单元
    - 一个Tablet增长到一定大小之后可以自动分裂成两个Tablets
    - 多个连续的Tablets可以合并成一个大的Tablet
    - 表块是被动态划分的（但相邻的应该在一起）

Big table存储

- Big table维护的是tablet和tablet server之间的映射，其以tablet为单位，实现分片，迁移等操作
- tablet采用的是range-based的分片方式，相近的row会被划分在同一个tablet里面：
    - range based对于范围查询是非常友好的，可以提高查询速度
    - 写入的时候流量会导入到同一个tablet，需要额外的split来达到均衡
- tablet内部采用了类似LSM Tree的存储方式，由一个memtable与多个sstable（sorted string table）组成：
    - 其中memtable是内存中的数据结构，而write ahead log、sstable则会持久化到GFS
    - 写操作时先写入日志文件，再从日志文件写入memtable：
        - 写日志时，通过Group Commit减少IPOS，提升写入性能
        - 利用两个写入线程避免GFS的写入时延毛刺
    - memtable是一个有序的字典，数据量到达一定情况下就会以sstable的形式写入到GFS
    - sstable是bigtable数据物理存储的基本单位，是顺序存储的：
        - 其内部包含多个block（64KB为单位），block index放在sstable末尾，打开sstable的时候block index会被加载到内存，二分查找block index就能找到需要的block，加速磁盘读取
        - 一个Locality Group是多个经常被一起访问的列族的组合
        - 同一个Locality Group中的数据会生成到同一个SSTable中
        - 可以将一个Locality Group配置成是否常驻内存的
    - 读操作需要先合并 memtable与SSTable中的数据

Big table系统模块

- **关键模块：**
    - Client Library
    - Master Server：
        - Tablet Server管理：
            - 监控探测Tablet服务器的增加和失效

        - 分配Tablet到Tablet Server
      - 负载均衡
      - 垃圾文件回收
      - 建表以及Schema变更管理（列族等变更）
    - Tablet Server：
      - 管理一系列的tablets，并负责它们的读写请求
      - tablets过大时进行切割操作
- **依赖服务：**
  - GFS 分布式文件系统 : 用来持久化存放Big Table的关联文件（日志文件和数据文件）
  - Chubby： 提供分布式的锁服务

Big table重要原理：

tablet定位： 使用三种层次结构存储信息（类似B+树）

- 首层将root tablet的位置信息以文件的形式存储在Chubby中，root tablet包含所有其他metadata tablet的位置信息
- 第二层metadata tablet保存着其他所有tablet的位置
- 第三层即为数据的tablet

注意：

- root tablet保存在表中的第一行，被特殊对待且永远不被切分，保证tablet的位置继承关系不会超过三层
- 客户端的lib缓存tablet的位置信息，这样可以不需要和master通信

集群成员变化与Tablet分配：

- **tablet server的加入和失效：**
  - Master利用Chubby探测Tablet server加入和离开的事件，每个tablet server在Chubby上由对应的唯一文件，启动时拿到该文件的互斥锁，master通过监听这些文件的父目录来检测tablet server的加入
  - teblet server可能由于网络波动断掉和chubby的通信，如果teblet 数据文件还在，server会一直重试获取独占锁，如果数据文件被移走，该服务器就会挂掉
  - master会定时询问tablet server的锁状态，如果被告知失去了锁或者没有回应，master会找Chubby要一个独占锁，如果有回应说明Chubby存活，此时：
    - tablet server可能挂掉了
    - tablet server无法与chubby通信
  - 确认到tablet服务器失效，master会删掉这个服务器注册的唯一文件，保证其不再提供服务，并将这个服务器的tablet标识为未分配状态

- **master失效与新master恢复：**
  - 要保证master与chubby之间的网络稳定，如果失去通信，master会挂掉
  - 新master的恢复过程：
    - 在 Chubby 上获取 Master 独有的锁，确保不会有另一个 Master 同时启动
    - 利用 Chubby 获取仍有效的 Tablet Server
    - 从各个 Tablet Server 处获取其所负责的 Tablet 列表，并向其表明自己作为新 Master 的身份，确保 Tablet Server 的后续通信能发往这个新 Master
    - Master 确保 Root Tablet 及 metadata 表的 Tablet 已完成分配
    - Master 扫描metadata表获取集群中的所有 Tablet，并对未分配的 Tablet 重新进行分配