# 大数据作业四

`homework`

## HBase是怎么基于HDFS，实现数据的随机读写的？并以数据插入及查找为例，说明具体的操作过程；

Hbase是Hadoop数据库，它是一个可以随机访问的、实时存储和检索大数据的平台，它的目标是在集群环境下支持大表的高性能访问；它是一个在HDFS上开发的面向列的分布式数据库，如果需要实时的随机访问超大规模数据集，就可以使用Hbase这一Hadoop应用；

Hbase的表可以有数十亿行，上百万列；每行都有一个可排序的主键和任意多的列，列可以根据需要动态的增加，同一张表中不同的行可以有截然不同的列；面向列(族)的存储和权限控制， 列(族)独立检索;对于空(null)的列，并不占用存储空间，表可以设计的非常稀疏；每个单元中的数据可以有多个版本， 默认情况下版本号自动分配，是单元格插入时的时间戳;Hbase中的数据都是字节数组，没有类型；Hbase表和RDBMS的表类似，单元格有版本，行是排序的，而只要有 列族预先存在，客户端可以随时把列添加到列族中去；

行由行键标识，行键可以是任意字节数组(目前支持最多64K，多数情况下10-100字节)；
表中数据都是根据行关键字进行排序的，排序使用的是字典顺序；
用户通过选择合适的行关键字，可以在数据访问时有效利用数据的位置相关性；

Hbase并不是简单地存储所有的列关键字，而是将其组织成所谓的列族(Column Family)；
列族是访问控制的基本单位，每个族中的数据都属于同一个类型，并且同族的数据会被压缩在一起保存；
Table中的每一行都有若干列,相同属性的列组成列族列族在HDFS中被对应保存为一个HFile；
一个列族可能有上百万个列
– 每一个列用(族名:限定符)(family:qualifier)来标识；
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，列限定符不必先创建；

Hbase表中每一个表项Cell都可以包含同一数据的多个版本，由时间戳来索引；

不同版本的表项内容按时间戳倒序排列，即最近的排序在前面；
Hbase的时间戳是64位整型，可以由Hbase来赋值，或者由用户应用程序来赋值；
客户端可以选择获取距离某个时间最近的版本，或者一次获取所有版本；
Hbase支持两种数据版本回收方式:每个数据单元，只存储指定个数的最新版本，保存指定时间长度的版本；

Table按(行关键字、列关键字、时间戳)可以定位到一个Cell(单元格)， Cell的内容本身是一个byte数组；

# Hbase 物理存储

• Table中的所有行都按照rowkey的字典序排列
• Table在行的方向上分割为多个Region

• Hbase中扩展和负载均衡的基本单元称为Region，Region本质上是以行键排序的连续存储区间
• Region按大小分割的，每个表开始只有一个region，随着数据增多，region不断增大，当增大到一个阀值的时候，region就会等分成两个新的region，之后会有越来越多的region
• 按照Hbase和现在的硬件能力，每台服务器的最佳加载Region数量差不多是10~1000，每个Region的最佳大小是1GB~2GB

• Region虽然是分布式存储的最小单元，但并不是底层物理存储的最小单元
– Region划分为若干Store进行存储，每个Store保存一个列族中的数据
– 每个Store又由一个memStore和多个StoreFile组成。StoreFile以HFile格式保存在HDFS

• Store到HFile
– Store由两部分组成，MemStore和StoreFile: MemStore是RegionServer上的一段内存空间；StoreFile是HDFS中的一个HFile文件
• 数据库操作会先存入MemStore，当MemStore满 了后会转存到StoreFile中
• 1个Store可包含多个StoreFile，并建立了 StoreFile索引

• HFile分为六个部分:
– Data Block:保存表中的数据，按key顺序存储,这部分可以被压缩。
– Meta Block(可选的):保存用户自定义的key/value对，可被压缩。

– File Info:HFile的元信息，不被压缩，用户可在此添加自己的元信息。
– Data Block Index:DataBlock的索引，定位要查找的数据在哪个块里。
– Meta Block Index(可选的):Meta Block的索引。
– Trailer:version, time range等。读取HFile时会首先读取Trailer，Trailer保存了每个段的起始位置。然后，DataBlockIndex会被读取到内存中，当检索某个key时，不需要扫描整个HFile，只需从内存中找到key所在的block，通过一次磁盘IO将整个block读取到内存中，再找到需要的key。

• HFile里面的每个Key/Value对就是一个简单的byte数组。但是这个byte数组里面包含了很多项，并且有固定的结构。

• HMaster服务器主要负责以下工作:
– 为Region服务器分配Regions
– 检测新加入的或者过期失效的Region服务器
– 对Region服务器进行负载均衡
– 对保存在HDFS上的文件进行垃圾收集
– 除此之外，它还处理对模式DDL(create,delete,etc)的相关操作，例如建立表和列族

• 每个HRegion服务器都管理一系列Region的集合
– 通常每个服务器有大约数十个至上千个Region
– 每个Region的尺寸大约是1GB到2GB
• 每个Region服务器负责处理它所加载的Region 的读写操作，以及在region过大时，对其进行分割
• 和很多Single-Master类型的分布式存储系统类似，客户端读写数据不经过Master服务器:客户程序直接和Region服务器通信进行读写操作

• HLog又称WAL(Write Ahead Log)，类似Mysql中的binlog，记录数据的所有变更，用来做灾难恢复
• 每个RegionServer维护一个Hlog,可以减少磁盘寻址次数，提高对表的写性能
• Hlog保存为Hadoop顺序文件(Sequence File)，顺序文件的Key是HLogKey对象，记录了写入数据的归属信息(表、Region名字、顺序号、写入时间等);HLog的Value是HBase的Key/Value对象， 即对应HFile中的Key/Value

# 数据插入

• Client启动一个操作来修改数据
• 每一个修改都被封装到KeyValue对象，并发送
到含有匹配Region的RegionServer
• Region检查数据是否与schema一致;
• 将更新写入HLog;
• 将更新写入Memstore;
• 判断MemStore的大小是否需要flush为StoreFile
• StoreFile达到一定的阈值后，按key进行合并

# 数据查找

1. 定位RS-找到.META.(-ROOT-表)
   – 根数据表，存放了.META.表的HRegionServer信息，存放在Zookeeper服务器
   – ROOT-表的Region不会被拆分，永远只有一个
   – 客户端首次访问获取-ROOT-表的位置并存入缓存
   – 行关键字:每个.META.表的Region索引
   – info:regioninfo:记录Region的一些必要信息
   – info:server:Region所在的RegionServer的地址和端口
   – info.serverstartcode:RegionServer对应.META.表持有进 程的启动时间
2. 定位RS-找到Region
   – 存储了所有表的元数据信息
   – 支持以表名和行关键字(或关键字的范围)查找到对应的RegionServer
   • 行关键字:表名、此Region起始关键字和Region的id
   – info:regioninfo:记录Region的一些必要信息
   – info:server:Region所在的RegionServer的地址和端口
   – infor.serverstartcode:RegionServer对应.META.表持有进程的启动时间