



# 大数据性能调优之HBase的RowKey设计

2016-01-19 软件开发那些事儿

免费订阅本平台！请点  上面蓝色字关注，查看精彩文章！

» 今日微信号力荐 ( 长按红色字复制 )  JAVA开发那些事儿

## 1 概述

HBase是一个分布式的、面向列的数据库，它和一般关系型数据库的最大区别是：HBase很适合于存储非结构化的数据，还有就是它基于列的而不是基于行的模式。

既然HBase是采用KeyValue的列存储，那Rowkey就是KeyValue的Key了，表示唯一——行。

Rowkey也是一段二进制码流，最大长度为64KB，内容可以由使用的用户自定义。数据加载时，一般也是根据Rowkey的二进制序由小到大进行的。

HBase是根据Rowkey来进行检索的，系统通过找到某个Rowkey (或者某个 Rowkey 范围)所在的Region，然后将查询数据的请求路由到该Region获取数据。HBase的检索支持3种方式：

- (1) 通过单个Rowkey访问，即按照某个Rowkey键值进行get操作，这样获取唯一——条记录；
- (2) 通过Rowkey的range进行scan，即通过设置startRowKey和endRowKey，在这个范围内进行扫描。这样可以按指定的条件获取一批记录；
- (3) 全表扫描，即直接扫描整张表中所有行记录。

HBASE按单个Rowkey检索的效率是很高的，耗时在1毫秒以下，每秒钟可获取1000~2000条记录，不过非key列的查询很慢。

## 2 HBase的RowKey设计

### 2.1 设计原则2.1.1 Rowkey长度原则

Rowkey是一个二进制码流，Rowkey的长度被很多开发者建议说设计在10~100个字节，不过建议是越短越好，不要超过16个字节。

原因如下：

- (1) 数据的持久化文件HFile中是按照KeyValue存储的，如果Rowkey过长比如100个字节，1000万列数据光Rowkey就要占用100\*1000万=10亿个字节，将近1G数据，这会极大影响HFile的存储效率；
- (2) MemStore将缓存部分数据到内存，如果Rowkey字段过长内存的有效利用率会降低，系统将无法缓存更多的数据，这会降低检索效率。因此Rowkey的字节长度越短越好。
- (3) 目前操作系统都是64位系统，内存8字节对齐。控制在16个字节，8字节的整数倍利用操作系统的最佳特性。

### 2.1.2 Rowkey散列原则

如果Rowkey是按时间戳的方式递增，不要将时间放在二进制码的前面，建议将Rowkey的高位作为散列字段，由程序循环生成，低位放时间字段，这样将提高数据均衡分布在每个Regionserver实现负载均衡的几率。如果没有散列字段，首字段直接是时间信息将产生所有新数据都在一个RegionServer上堆积的热点现象，这样在做数据检索的时候负载将会集中在个别RegionServer，降低查询效率。

### 2.1.3 Rowkey唯一原则

必须在设计上保证其唯一性。

2.2 应用场景

基于Rowkey的上述3个原则，应对不同应用场景有不同的Rowkey设计建议。

2.2.1 针对事务数据Rowkey设计

事务数据是带时间属性的，建议将时间信息存入到Rowkey中，这有助于提示查询检索速度。对于事务数据建议缺省就按天为数据建表，这样设计的好处是多方面的。按天分表后，时间信息就可以去掉日期部分只保留小时分钟毫秒，这样4个字节即可搞定。加上散列字段2个字节一共6个字节即可组成唯一 Rowkey。如下图所示：

| 事务数据Rowkey设计           |      |                                   |      |      |      |      |
|------------------------|------|-----------------------------------|------|------|------|------|
| 第0字节                   | 第1字节 | 第2字节                              | 第3字节 | 第4字节 | 第5字节 | ...  |
| 散列字段                   |      | 时间字段(毫秒)                          |      |      |      | 扩展字段 |
| 0~65535(0x0000~0xFFFF) |      | 0~86399999(0x00000000~0x05265BFF) |      |      |      |      |

这样的设计从操作系统内存管理层面无法节省开销，因为64位操作系统是必须8字节对齐。但是对于持久化存储中Rowkey部分可以节省25%的开销。也许有人要问为什么不将时间字段以主机字节序保存，这样它也可以作为散列字段了。这是因为时间范围内的数据还是尽量保证连续，相同时间范围内的数据查找的概率很大，对查询检索有好的效果，因此使用独立的散列字段效果更好，对于某些应用，我们可以考虑利用散列字段全部或者部分来存储某些数据的字段信息，只要保证相同散列值在同一时间（毫秒）唯一。

2.2.2 针对统计数据的Rowkey设计

统计数据也是带时间属性的，统计数据最小单位只会到分钟（到秒预统计就没意义了）。同时对于统计数据我们也缺省采用按天数据分表，这样设计的好处无需多说。按天分表后，时间信息只需要保留小时分钟，那么0~1400只需占用两个字节即可保存时间信息。由于统计数据某些维度数量非常庞大，因此需要4个字节作为序列字段，因此将散列字段同时作为序列字段使用也是6个字节组成唯一Rowkey。如下图所示：

| 统计数据Rowkey设计           |      |      |      |                       |      |      |
|------------------------|------|------|------|-----------------------|------|------|
| 第0字节                   | 第1字节 | 第2字节 | 第3字节 | 第4字节                  | 第5字节 | ...  |
| 散列字段(序列字段)             |      |      |      | 时间字段(分钟)              |      | 扩展字段 |
| 0x00000000~0xFFFFFFFF) |      |      |      | 0~1439(0x0000~0x059F) |      |      |

同样这样的设计从操作系统内存管理层面无法节省开销，因为64位操作系统是必须8字节对齐。但是对于持久化存储中Rowkey部分可以节省25%的开销。预统计数据可能涉及到多次反复的重计算要求，需确保作废的数据能有效删除，同时不能影响散列的均衡效果，因此要特殊处理。

2.2.3 针对通用数据的Rowkey设计

通用数据采用自增序列作为唯一主键，用户可以选择按天建分表也可以选择单表模式。这种模式需要确保同时多个入库加载模块运行时散列字段（序列字段）的唯一性。可以考虑给不同的加载模块赋予唯一因子区别。设计结构如下图所示。

|                        |      |      |      |                |
|------------------------|------|------|------|----------------|
| 通用数据Rowkey设计           |      |      |      |                |
| 第0字节                   | 第1字节 | 第2字节 | 第3字节 | ...            |
| 散列字段(序列字段 )            |      |      |      | 扩展字段（控制在12字节内） |
| 0x00000000~0xFFFFFFFF) |      |      |      | 可由多个用户字段组成     |

2.2.4 支持多条件查询的RowKey设计

HBase按指定的条件获取一批记录时，使用的就是scan方法。 scan方法有以下特点：

- （1）scan可以通过setCaching与setBatch方法提高速度（以空间换时间）；
- （2）scan可以通过setStartRow与setEndRow来限定范围。范围越小，性能越高。

通过巧妙的RowKey设计使我们批量获取记录集合中的元素挨在一起（应该在同一个Region下），可以在遍历结果时获得很好的性能。

- （3）scan可以通过setFilter方法添加过滤器，这也是分页、多条件查询的基础。

在满足长度、三列、唯一原则后，我们需要考虑如何通过巧妙设计RowKey以利用scan方法的范围功能，使得获取一批记录的查询速度能提高。下例就描述如何将多个列组合成一个RowKey，使用scan的range来达到较快查询速度。

例子：

我们在表中存储的是文件信息，每个文件有5个属性：文件id（long，全局唯一）、创建时间（long）、文件名（String）、分类名（String）、所有者（User）。

我们可以输入的查询条件：文件创建时间区间（比如从20120901到20120914期间创建的文件），文件名（“中国好声音”），分类（“综艺”），所有者（“浙江卫视”）。

假设当前我们一共有如下文件：

| ID | CreateTime | Name      | Category | UserID |
|----|------------|-----------|----------|--------|
| 1  | 20120902   | 中国好声音第1期  | 综艺       | 1      |
| 2  | 20120904   | 中国好声音第2期  | 综艺       | 1      |
| 3  | 20120906   | 中国好声音外卡赛  | 综艺       | 1      |
| 4  | 20120908   | 中国好声音第3期  | 综艺       | 1      |
| 5  | 20120910   | 中国好声音第4期  | 综艺       | 1      |
| 6  | 20120912   | 中国好声音选手采访 | 综艺花絮     | 2      |
| 7  | 20120914   | 中国好声音第5期  | 综艺       | 1      |
| 8  | 20120916   | 中国好声音录制花絮 | 综艺花絮     | 2      |
| 9  | 20120918   | 张玮独家专访    | 花絮       | 3      |
| 10 | 20120920   | 加多宝凉茶广告   | 综艺广告     | 4      |

这里UserID应该对应另一张User表，暂不列出。我们只需知道UserID的含义：

1代表 浙江卫视； 2代表 好声音剧组； 3代表 XX微博； 4代表赞助商。调用查询接口的时候将上述5个条件同时输入find(20120901,20121001,"中国好声音","综艺","浙江卫视")。此时我们应该得到

记录应该有第1、2、3、4、5、7条。第6条由于不属于“浙江卫视”应该不被选中。我们在设计RowKey时可以这样做：采用 UserID + CreateTime + FileID组成RowKey，这样既能满足多条件查询，又能有很快的查询速度。

需要注意以下几点：

（1）每条记录的RowKey，每个字段都需要填充到相同长度。假如预期我们最多有10万量级的用户，则userID应该统一填充至6位，如000001，000002...

（2）结尾添加全局唯一的FileID的用意也是使每个文件对应的记录全局唯一。避免当UserID与CreateTime相同时的两个不同文件记录相互覆盖。

按照这种RowKey存储上述文件记录，在HBase表中是下面的结构：

rowKey ( userID 6 + time 8 + fileID 6 ) name category ....

00000120120902000001

00000120120904000002

00000120120906000003

00000120120908000004

00000120120910000005

00000120120914000007

00000220120912000006

00000220120916000008

00000320120918000009

00000420120920000010

怎样用这张表？

在建立一个scan对象后，我们setStartRow(00000120120901)，setEndRow(00000120120914)。这样，scan时只扫描userID=1的数据，且时间范围限定在这个指定的时间段内，满足了按用户以及按时间范围对结果的筛选。并且由于记录集中存储，性能很好。

然后使用

SingleColumnValueFilter ( org.apache.hadoop.hbase.filter.SingleColumnValueFilter )，共4个，分别约束name的上下限，与category的上下限。满足按同时按文件名以及分类名的前缀匹配。

（注意：使用SingleColumnValueFilter会影响查询性能，在真正处理海量数据时会消耗很大的资源，且需要较长的时间）

如果需要分页还可以再加一个PageFilter限制返回记录的个数。

以上，我们完成了高性能的支持多条件查询的HBase表结构设计。

## 关注软件开发哪些事儿(java,大数据挖掘)



请使用微信扫描二维码关注此公众号

分享java,大数据,数据挖掘,关系数据库,NoSQL数据库,SOA,缓存,消息队列,Linux运维等视频教程,电子文档,实战项目源码,开发经验。机器人“热咖啡”能回答你java笔试面试题目,助你实现高新软件工程师梦想!



软件开发那些事儿

[阅读原文](#)