[**Hive-ORC文件存储格式**](http://www.cnblogs.com/cxzdy/p/5910760.html)

ORC文件格式是从Hive-0.11版本开始的。关于ORC文件格式的[官方文档](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC)，以及基于[官方文档的翻译](http://www.iteblog.com/archives/1014#HiveORCFile)内容这里就不赘述了，有兴趣的可以仔细研究了解一下。本文接下来根据论文《Major Technical Advancements in Apache Hive》中的内容进行深入的研究。

**一、ORC文件格式**

ORC的全称是(Optimized Record Columnar)，使用ORC文件格式可以提高hive读、写和处理数据的能力。ORC在RCFile的基础上进行了一定的改进，所以与RCFile相比，具有以下一些优势：

- 1、ORC中的特定的序列化与反序列化操作可以使ORC file writer根据数据类型进行写出。

- 2、提供了多种RCFile中没有的indexes，这些indexes可以使ORC的reader很快的读到需要的数据，并且跳过无用数据，这使得ORC文件中的数据可以很快的得到访问。

- 3、由于ORC file writer可以根据数据类型进行写出，所以ORC可以支持复杂的数据结构（比如Map等）。

- 4、除了上面三个理论上就具有的优势之外，ORC的具体实现上还有一些其他的优势，比如ORC的stripe默认大小更大，为ORC writer提供了一个memory manager来管理内存使用情况。

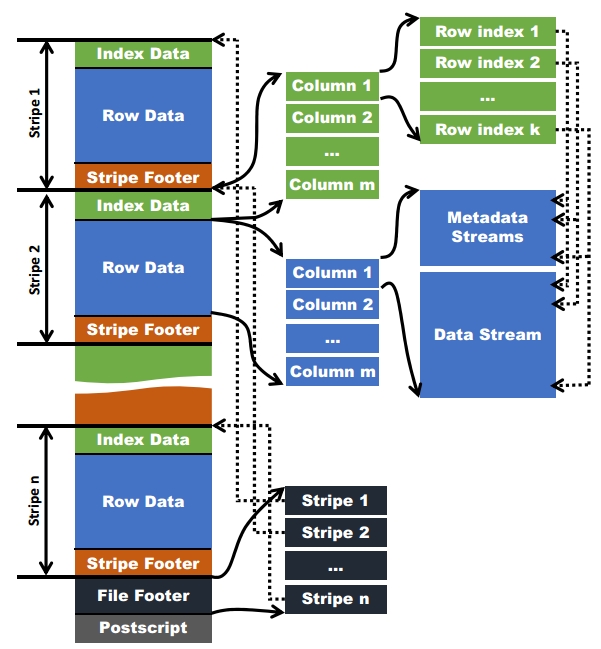


图1-ORC文件结构图

**二、ORC数据存储方法**

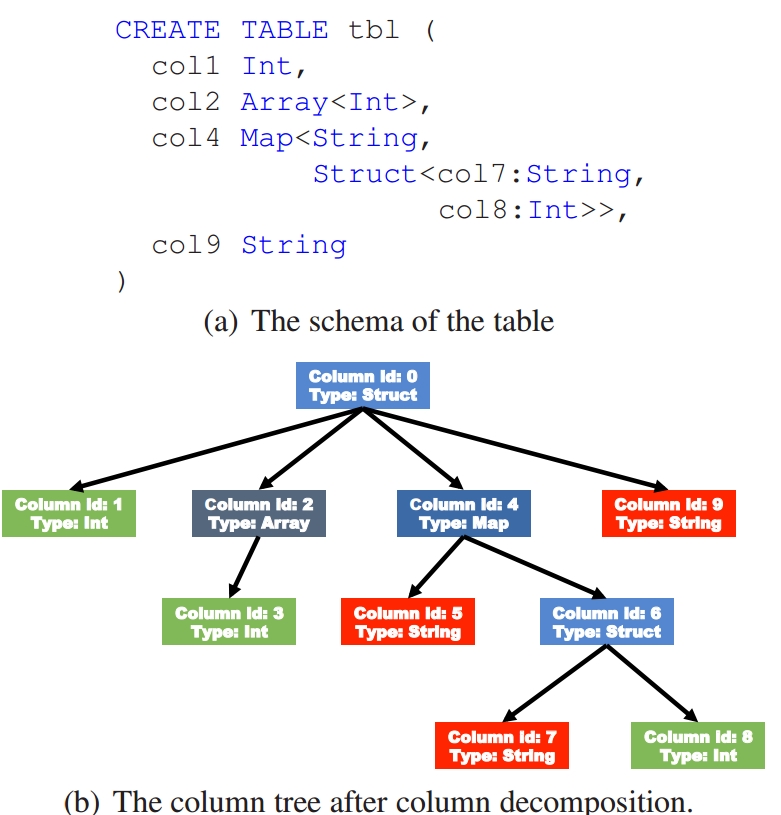
在ORC格式的hive表中，记录首先会被横向的切分为多个**stripes**，然后在每一个stripe内数据以列为单位进行存储，所有列的内容都保存在同一个文件中。每个stripe的默认大小为256MB，相对于RCFile每个4MB的stripe而言，更大的stripe使ORC的数据读取更加高效。

对于复杂数据类型，比如Map，ORC文件会将一个复杂数据类型字段解析成多个子字段。下表中列举了ORC文件中对于复杂数据类型的解析

|  |  |
| --- | --- |
| **Data type** | **Chile columns** |
| Array | 一个包含所有数组元素的子字段 |
| Map | 两个子字段，一个key字段，一个value字段 |
| Struct | 每一个属性对应一个子字段 |
| Union | 每一个属性对应一个子字段 |

当字段类型都被解析后，会由这些字段类型组成一个字段树，只有树的叶子节点才会保存表数据，这些叶子节点中的数据形成一个数据流，如上图中的Data Stream。

为了使ORC文件的reader更加高效的读取数据，字段的metadata会保存在Meta Stream中。在字段树中，每一个非叶子节点记录的就是字段的metadata，比如对一个array来说，会记录它的长度。下图根据表的字段类型生成了一个对应的字段树。



图二－字段树结构图

在Hive-0.13中，ORC文件格式只支持读取指定字段，还不支持只读取特殊字段类型中的指定部分。

使用ORC文件格式时，用户可以使用HDFS的每一个block存储ORC文件的一个stripe。对于一个ORC文件来说，stripe的大小一般需要设置得比HDFS的block小，如果不这样的话，一个stripe就会分别在HDFS的多个block上，当读取这种数据时就会发生远程读数据的行为。如果设置stripe的只保存在一个block上的话，如果当前block上的剩余空间不足以存储下一个strpie，ORC的writer接下来会将数据打散保存在block剩余的空间上，直到这个block存满为止。这样，下一个stripe又会从下一个block开始存储。

**三、索引**

在ORC文件中添加索引是为了更加高效的从HDFS读取数据。在ORC文件中使用的是稀疏索引(sparse indexes)。在ORC文件中主要有两种用途的索引，一个是数据统计(Data Statistics)索引，一个是位置指针(Position Pointers)索引。

**1. Data Statistics**

ORC reader用这个索引来跳过读取不必要的数据，在ORC writer生成ORC文件时会创建这个索引文件。这个索引中统计的信息主要有记录的条数，记录的max, min, sum值，以及对text类型和binary类型字段还会记录其长度。对于复杂数据类型，比如Array, Map, Struct, Union，它们的子字段中也会记录这些统计信息。

在ORC文件中，Data Statistics有三个level。

1. **file level statistics**

在ORC文件的末尾会记录文件级别的统计信息，会记录整个文件中columns的统计信息。这些信息主要用于查询的优化，也可以为一些简单的聚合查询比如max, min, sum输出结果。

1. **stripe level statistics**

ORC文件会保存每个字段stripe级别的统计信息，ORC reader使用这些统计信息来确定对于一个查询语句来说，需要读入哪些stripe中的记录。比如说某个stripe的字段max(a)=10，min(a)=3，那么当where条件为a >10或者a <3时，那么这个stripe中的所有记录在查询语句执行时不会被读入。

1. **index group level statistics**

为了进一步的避免读入不必要的数据，在逻辑上将一个column的index以一个给定的值(默认为10000，可由参数配置)分割为多个index组。以10000条记录为一个组，对数据进行统计。Hive查询引擎会将where条件中的约束传递给ORC reader，这些reader根据组级别的统计信息，过滤掉不必要的数据。如果该值设置的太小，就会保存更多的统计信息，用户需要根据自己数据的特点权衡一个合理的值。

**3. Position Pointers**

当读取一个ORC文件时，ORC reader需要有两个位置信息才能准确的进行数据读取操作。

1. **metadata streams和data streams中每个group的开始位置**

由于每个stripe中有多个group，ORC reader需要知道每个group的metadata streams和data streams的开始位置。图１中右边的虚线代表的就是这种pointer。

1. **stripes的开始位置**

由于一个ORC文件可以包含多个stripes，并且一个HDFS block也能包含多个stripes。为了快速定位指定stripe的位置，需要知道每个stripe的开始位置。这些信息会保存在ORC file的File Footer中。如图1中间位置的虚线所示。

**四、文件压缩**

ORC文件使用两级压缩机制，首先将一个数据流使用流式编码器进行编码，然后使用一个可选的压缩器对数据流进行进一步压缩。

一个column可能保存在一个或多个数据流中，可以将数据流划分为以下四种类型：

**• Byte Stream**

字节流保存一系列的字节数据，不对数据进行编码。

**• Run Length Byte Stream**

字节长度字节流保存一系列的字节数据，对于相同的字节，保存这个重复值以及该值在字节流中出现的位置。

**• Integer Stream**

整形数据流保存一系列整形数据。可以对数据量进行字节长度编码以及delta编码。具体使用哪种编码方式需要根据整形流中的子序列模式来确定。

**• Bit Field Stream**

比特流主要用来保存boolean值组成的序列，一个字节代表一个boolean值，在比特流的底层是用Run Length Byte Stream来实现的。

接下来会以Integer和String类型的字段举例来说明。

1. **Integer**

对于一个整形字段，会同时使用一个比特流和整形流。比特流用于标识某个值是否为null，整形流用于保存该整形字段非空记录的整数值。

1. **String**

对于一个String类型字段，ORC writer在开始时会检查该字段值中不同的内容数占非空记录总数的百分比不超过0.8的话，就使用字典编码，字段值会保存在一个比特流，一个字节流及两个整形流中。比特流也是用于标识null值的，字节流用于存储字典值，一个整形流用于存储字典中每个词条的长度，另一个整形流用于记录字段值。

如果不能用字典编码，ORC writer会知道这个字段的重复值太少，用字典编码效率不高，ORC writer会使用一个字节流保存String字段的值，然后用一个整形流来保存每个字段的字节长度。

在ORC文件中，在各种数据流的底层，用户可以自选ZLIB, Snappy和LZO压缩方式对数据流进行压缩。编码器一般会将一个数据流压缩成一个个小的压缩单元，在目前的实现中，压缩单元的默认大小是256KB。

**五、内存管理**

当ORC writer写数据时，会将整个stripe保存在内存中。由于stripe的默认值一般比较大，当有多个ORC writer同时写数据时，可能会导致内存不足。为了现在这种并发写时的内存消耗，ORC文件中引入了一个内存管理器。在一个Map或者Reduce任务中内存管理器会设置一个阈值，这个阈值会限制writer使用的总内存大小。当有新的writer需要写出数据时，会向内存管理器注册其大小（一般也就是stripe的大小），当内存管理器接收到的总注册大小超过阈值时，内存管理器会将stripe的实际大小按该writer注册的内存大小与总注册内存大小的比例进行缩小。当有writer关闭时，内存管理器会将其注册的内存从总注册内存中注销。

**六、参数**

|  |  |  |
| --- | --- | --- |
| **参数名** | **默认值** | **说明** |
| hive.exec.orc.default.stripe.size | 256\*1024\*1024 | stripe的默认大小 |
| hive.exec.orc.default.block.size | 256\*1024\*1024 | orc文件在文件系统中的默认block大小，从hive-0.14开始 |
| hive.exec.orc.dictionary.key.size.threshold | 0.8 | String类型字段使用字典编码的阈值 |
| hive.exec.orc.default.row.index.stride | 10000 | stripe中的分组大小 |
| hive.exec.orc.default.compress | ZLIB | ORC文件的默认压缩方式 |
| hive.exec.orc.skip.corrupt.data | false | 遇到错误数据的处理方式，false直接抛出异常，true则跳过该记录 |