CarbonData源码浅析一：Create Table

???? ?? ?????????https://blog.csdn.net/zzcclp/article/details/80161130

前言

一个偶然的机会，从某Spark微信群知道了CarbonData，从断断续续地去了解，到测试 1.2 版本，再到实际应用 1.3 版本的流式入库，也一年有余，在这期间，得到了 CarbonData 社区的陈亮，李昆，蔡强等大牛的鼎力支持，自己也从认识CarbonData 到应用 CarbonData，再到参与社区的转变，感谢他们！

要把CarbonData用得好，姿势必须正确:)，优化步骤就必不可少，熟悉源码的必要性就不言而喻了，因此准备再进一步研究学习CarbonData源码，同时把学习中的一些点滴记录下来。目前，暂定推出四篇博客，从 Create Table，Load Data（DataFrame.write），Select Data，及结合流式入库等四个方面来浅析下 CarbonData 源码。

言归正传哈。

简介

CarbonData 是首个由中国公司发起并捐献给 Apache 基金会的开源项目，于2017年4月正式成为 Apache 顶级项目，由华为开源并支持 Hadoop 的高性能列式存储文件格式，其目的是提供一种统一的数据存储方案，以一份数据同时支持大数据分析的多种应用场景，All In One，并通过多级索引、字典编码、列式存储等特性提升 I/O 扫描和计算性能，实现百亿数据级秒级响应。CarbonData 里程碑版1.3于2018年2月正式发布，该版本包含了集成 Spark 2.2.1，支持预聚合，流式准实时入库，支持标准Hive分区等几个重要特性，而即将发布的1.4版本更是包含了支持 Lucene Index 加强文本检索能力，支持 Bloom Filter Index 等重要特性，在 All In One 的道路上又迈进了一步。

目前 CarbonData 与 Spark，Presto, Hive 等框架做了集成，其中与Spark的集成最深入，提供了基于索引、预聚合、全局字典等更多查询优化手段来提升性能，也提供了数据更新、删除、增量入库等数据管理能力，可以说是在 Spark 开源框架上针对数据仓库类应用的增强。一个计算框架方面的 All In One，一个存储格式方面的 All In One（其实 CarbonData 的功能已经远远超越了数据格式的范畴了），两者结合碰出的火花着实十分吸引人，这也是当初我会想要使用CarbonData 并深入了解它的最根本原因。

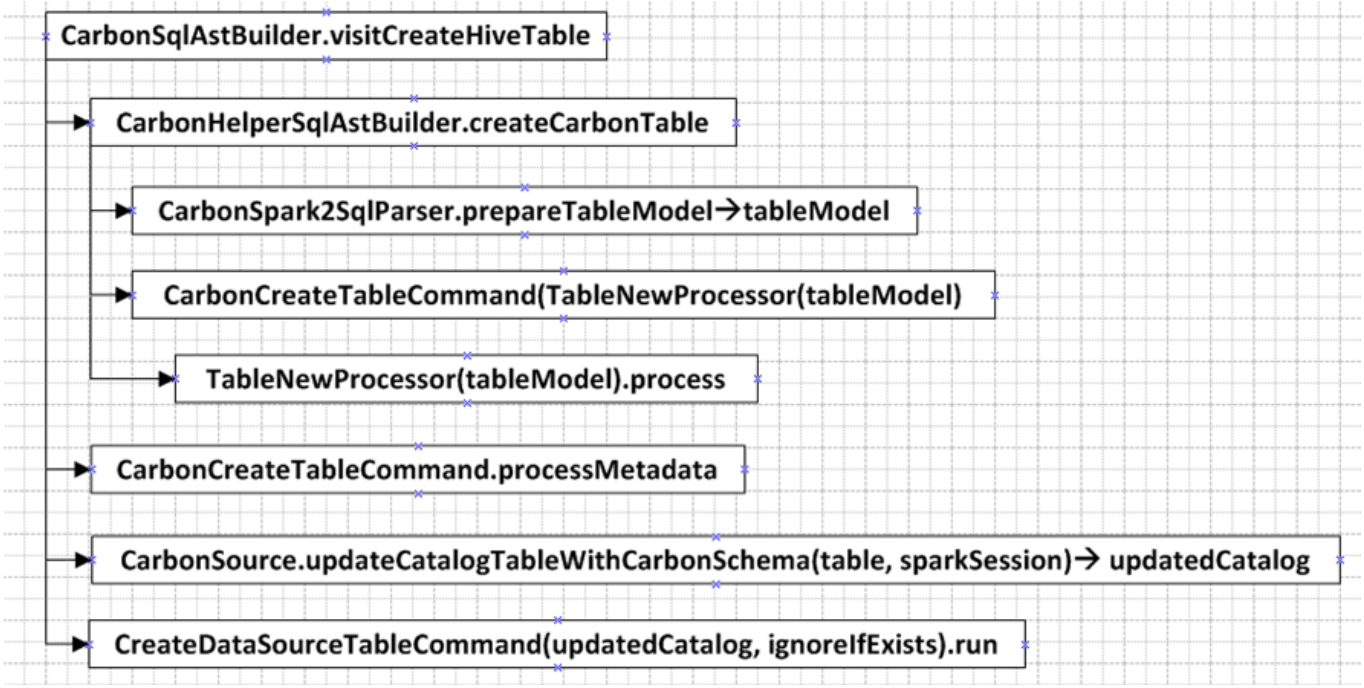
此系列就是基于 CarbonData 里程碑版1.3的源码进行浅析。

Create Table浅析

建表语句兼容 SparkSQL，只是使用 'carbondata' 这个DataSource，并扩展了一些属性来描述 Global Dictionary Columns，Sort Columns，Sort Scope 等：

- DICTIONARY_INCLUDE：指定做全局字典的列，主要用途在于：
    1. 压缩数据，String 类型转换为 Int 进行存储，并采用RLE算法进行压缩，因为压缩率提升了，而且是全局统一编码的字典，所以在做group by汇聚计算时读取数据量和shuffle的数据量减少了很多，带来性能提升；
    2. 1.3版本前，默认是对所有String类型的列做全局字典，不需要的列需要使用DICTIONARY_EXCLUDE属性来排除，对于上百列String类型列的表，配置起来有点麻烦，而且上百列String列做全局字典对导入也是个梦魇，因此在1.3版本，废除了String类型列默认做全局字典的规则，只保留了DICTIONARY_INCLUDE来配置需要做全局字典的列，其他列一律不做；
- SORT_COLUMNS：指定索引列，CarbonData中默认采用一种多级索引的策略，能在Driver侧做Pruning时过滤掉不必要的Block或Blocklet（文件内的数据块）：
    1. 提升过滤查询的性能，对于过滤查询、点查，设置合理的SORT_COLUMNS，会有不小的性能提升；
    2. 多级索引的顺序按照SORT_COLUMNS配置的列顺序，越常用的查询列放在最前面，相同查询频率的列按基数从小到大排列；
    3. 如果配置SORT_COLUMNS=""，即不做索引；
- NO_INVERTED_INDEX：SORT_COLUMNS配置的列中，如果不做倒排索引，可以通过该属性进行排除，因为做倒排索引会使文件变大，如果希望提升压缩率，可以减少建立倒排索引的列；
- SORT_SCOPE：加载时，数据排序的范围，目前支持如下几种：
    1. LOCAL_SORT：默认值，表示在一个node下做数据排序；
    2. NO_SORT：即不排序，在需要快速入库时使用，可以在入库后系统闲时通过Compaction命令再建立索引；
    3. BATCH_SORT：表示在一个node下，内存排序后直接生成carbondata文件，不再进行node下的全排序；使用该配置，可以提升加载速度，但查询性能不如LOCAL_SORT；
    4. GLOBAL_SORT：使用spark的资源调度算法和GroupBy做数据排序，会做shuffle操作，因此对于点查，会有不错的性能提升，但加载性能会不如LOCAL_SORT；
- TABLE_BLOCKSIZE：表的block大小，默认值是1024MB，类似于HDFS中的block概念，对每条数据的size比较小的表做点查，可以设置较小的值，达到性能的提升
- STREAMING：设置为true即表示启动Spark流式入库作业进行小批量入库，为了避免小文件问题，CarbonData在流式入库时会先把数据append到一个行存文件中，在文件达到一定大小后再转换为列存文件；

建表流程



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

下面对建表流程进行详细的分析。

CarbonHelperSqlAstBuilder.createCarbonTable：

- 生成TableIdentifier；
- 对不支持的建表语句进行验证，比如不支持Temp View，SKEWED BY，CLUSTERED BY及EXTERNAL TABLE；
- 获取column列表，如果有分区字段，则进行合并àfields；
- 如果是streaming表，暂不支持分区表；

CarbonSpark2SqlParser.prepareTableModel：

该方法就是对所有列进行dimension及measure的划分

- 对所有列fields按建表时的顺序进行编号；
- 调用extractDimAndMsrFields方法开始划分dimension和measure列：

1. sortcolumns目前暂不支持如下类型："array", "struct","double", "float", "decimal"，希望后续有更多的人来参与实现；
2. 如果在DICTIONARY_INCLUDE中有定义的列，则加到dimFields；
3. 为TIMESTAMP类型且不存在于dictIncludeCols中的列，则同时加到noDictionaryDims和dimFields；
4. 类型如果是"string", "array","struct", "timestamp", "date", "char"，则加到dimFields，但如果是string类型，则同时需要加入到noDictionaryDims；

5. 如果在SORT_COLUMNS中的列，则同时加入到noDictionaryDims和dimFields；
6. 其他则加入到msrFields；
7. 如果没有定义sort columns，则把dimFields中（除"array", "struct"类型外）的列都作为sortKeyDims；

- 调用extractNoInvertedIndexColumns获取NO_INVERTED_INDEX的列信息；
- 调用getPartitionInfo获取分区信息：支持HASH，RANGE，LIST三种分区类型（CarbonData社区用户主要用的是Hive标准分区，HASH，RANGE，LIST三种分区当前为Alpha特性）；
- 检验TABLE_BLOCKSIZE，只支持1-2048MB的范围；
- 检验TableLevelCompaction属性；
- 对dimFields中列进行重新排序，把复杂（Array，Struct）类型的数据排到最后，普通类型放在前面；

TableNewProcessor(tableModel).process：

- 对TableModel中的sortKeyDims、dimCols、msrCols进行UUID和Encoding设置，生成对应的ColumnSchema：

1. 对于sortKeyDims：出现在noDictionaryDims中的列，都不具有Encoding.DICTIONARY；Date和Timestamp（不在noDictionaryDims中）类型则具有Encoding.DIRECT_DICTIONARY；
2. 对于dimCols：不在sortKeyDims中的列都增加Encoding.DICTIONARY；
3. 对于msrCols：不具有任何Encoding；

- 扫描allColumns，如果列在TableModel.noInvertedIdxCols中或者TableModel.msrCols中，则为NO INVERTED INDEX
  列，否则就具有Encoding.INVERTED_INDEX
  （此部分逻辑可以进一步优化，后续会提交PR到社区，应该是SORT COLUMNS列都默认具有INVERTED INDEX，除非在NO_INVERTED_INDEX属性中特别指定的列，不过该问题不会影响实际的写数据，写数据时的判断逻辑是正确的）；
- 生成val tableInfo = new TableInfo()，val tableSchema = new TableSchema()，tableInfo.setFactTable(tableSchema)；
- 至此，列的划分结束，allColumns中的列是按照先sort column列，非sort column的dimensions列、complex data type 列、measures列的顺序排序；

CarbonCreateTableCommand.processMetadata：

- 生成CarbonTable carbonTable =CarbonTable.buildFromTableInfo(tableInfo)；
- 调用CarbonUtil.convertToMultiGsonStrings，用Gson把TableInfo转为json string，并分割为几个部分，中间会加上一些carbonSchemaPartsNo及carbonSchemaN的字符串，每个部分长度为4000；
- 拼凑实际的Create Table语句，指定DataSource：'USING org.apache.spark.sql.CarbonS

ource'，并把上一步骤分割的字符串作为建表的options加入到SQL中；

>

```
sparkSession.sql(
  s"""CREATE TABLE $dbName.$tableName
     |(${ rawSchema })
     |USING org.apache.spark.sql.CarbonSource
     |OPTIONS (
     |  tableName "$tableName",
     |  dbName "$dbName",
     |  tablePath "$tablePath",
     |  path "$tablePath",
     |  isVisible "$isVisible"
     |  $carbonSchemaString)
     |  $partitionString
  """.stripMargin)
```

如果想
及时了解Spar
k、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

- CarbonSource.updateCatalogTableWithCarbonSchema：把上步骤中建表语句中的carbonSchemaPartsNo和carbonSchemaN属性读取出来拼凑为TableInfo的json string，然后反序列化为TableInfo实例，调用CarbonFileMetastore.saveToDisk(tableInfo, properties("tablePath"))方法，使用ThriftWriter写到Metadata/schema文件中；然后把建表语句中的carbonSchemaPartsNo和carbonSchemaN属性去除；
- CreateDataSourceTableCommand.run：调用sessionState.catalog.createTable(newTable,ignoreIfExists = false)执行建表语句；