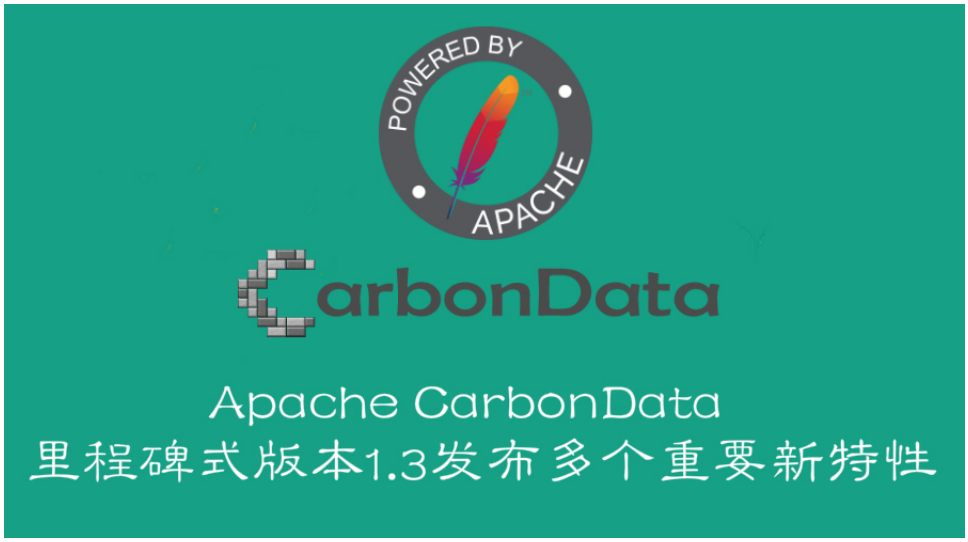
Apache CarbonData里程碑式版本1.3发布，多个重要新特性

CarbonData是一种高性能大数据存储方案，支持快速过滤查找和即席OLAP分析，已在20+企业生产环境上部署应用，其中最大的单一集群数据规模达到几万亿。针对当前大数据领域分析场景需求各异而导致的存储冗余问题，业务驱动下的数据分析灵活性要求越来越高，CarbonData提供了一种新的融合数据存储方案，以一份数据同时支持多种应用场景，并通过多级索引、字典编码、预聚合、动态Partition、准实时数据查询等特性提升了IO扫描和计算性能，实现万亿数据分析秒级响应。



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

我们来看下，CarbonData 1.3.0有哪些重大特性：

支持与Spark 2.2.1集成

CarbonData 1.3.0支持与最新Spark稳定版Spark 2.2.1版本集成。

支持预聚合，灵活高性能多维分析，无需预先数据建模

在1.3.0中，CarbonData的预聚合特性，与传统BI系统的CUBE方案最大区别是，用户不需要预先进行Cube建模以及修改任何SQL语句，既可加速OLAP分析性能，又可查询明细数据，做到一份数据满足多种应用场景。具体的用法如下：

a) 创建主表：

```
CREATE TABLE sales (
order_time TIMESTAMP,
```

```
user_id STRING,
sex STRING,
country STRING,
quantity INT,
price BIGINT)
STORED BY 'carbondata'
```

b) 基于上面主表sales创建预聚合表：

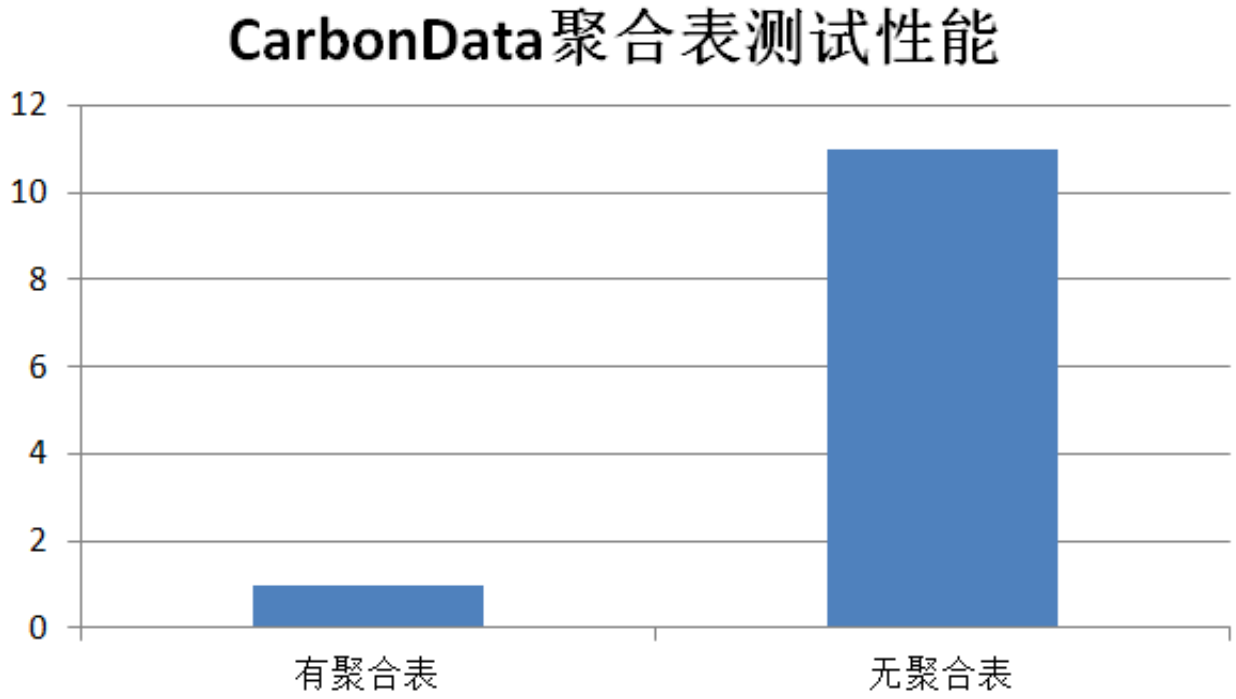
```
CREATE DATAMAP agg_sales
ON TABLE sales
USING "preaggregate"
AS
SELECT country, sex, sum(quantity), avg(price)
FROM sales
GROUP BY country, sex
```

c) 当用户做查询时，首先用户不需要改SQL查询语句，仍然使用主表进行查询。当收到查询请求后，CarbonData会对SQL做基于代价的自动优化，将SQL改写为预聚合表上的查询，例如下列语句会命中预聚合表，显著提升查询性能：

```
SELECT country, sex, sum(quantity), avg(price) FROM sales GROUP BY country, sex；
// 命中，完全和聚合表一样
SELECT sex, sum(quantity) FROM sales GROUP BY sex；//命中，聚合表的部分查询
SELECT country, avg(price) FROM sales GROUP BY country；//命中，聚合表的部分查询
SELECT country, sum(price) FROM sales GROUP BY country；
//命中，因为聚合表里avg(price)是通过sum(price)/count(price)产生，所以sum(price)也命中
SELECT sex, avg(quantity) FROM sales GROUP BY sex; //没命中，需要创建新的预聚合表
SELECT max(price), country FROM sales GROUP BY country；//没命中，需要创建新的预聚合表
SELECT user_id, country, sex, sum(quantity), avg(price) FROM sales GROUP BY user_id, country,
 sex;
//没命中，需要创建新的预聚合表
```

d) 在1.3.0版本中，支持的预聚合表达式有：SUM、AVG、MAX、MIN、COUNT

e) 在1亿数据量上实测性能可提升10+倍以上，数据越大，性能提升效果越好。大家可以参考例子：/apache/carbondata/examples/PreAggregateTableExample.scala



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

针对时间序列数据内置时间维的预聚合，支持自动上卷

对于时间序列数据，CarbonData内置了对时间维度的理解，为用户提供了易用语法创建预聚合表。具体用法如下：

a) 创建主表：

```
CREATE TABLE sales (
order_time TIMESTAMP,
user_id STRING,
sex STRING,
country STRING,
quantity INT,
price BIGINT)
STORED BY 'carbondata'
```

b) 分别创建Year、Month、Day、Hour、Minute粒度的聚合表：

```
//创建年粒度聚合表
CREATE DATAMAP agg_year
```

```
ON TABLE sales
USING "timeseries"
DMPROPERTIES (
'event_time’=’order_time’,
'year_granualrity’=’1’) AS
SELECT order_time, country, sex, sum(quantity), max(quantity),
count(user_id), sum(price),avg(price)
FROM sales GROUP BY order_time, country, sex


//创建月粒度聚合表
CREATE DATAMAP agg_month
ON TABLE sales
USING "timeseries"
DMPROPERTIES (
'event_time’=’order_time’,
'month_granualrity’=’1’) AS
SELECT order_time, country, sex, sum(quantity), max(quantity),
count(user_id), sum(price),avg(price)
FROM sales GROUP BY order_time, country, sex


//创建天粒度聚合表
CREATE DATAMAP agg_day
ON TABLE sales
USING "timeseries"
DMPROPERTIES (
'event_time’=’order_time’,
'day_granualrity’=’1’) AS
SELECT order_time, country, sex, sum(quantity), max(quantity),
count(user_id), sum(price),avg(price)
FROM sales GROUP BY order_time, country, sex


//创建小时粒度聚合表
CREATE DATAMAP agg_sales_hour
ON TABLE sales
USING "timeseries"
DMPROPERTIES (
'event_time’=’order_time’,
'hour_granualrity’=’1’) AS
SELECT order_time, country, sex, sum(quantity), max(quantity), count(user_id),
sum(price),avg(price)
FROM sales GROUP BY order_time, country, sex
```

```
//创建分钟粒度聚合表
CREATE DATAMAP agg_minute
ON TABLE sales
USING "timeseries"
DMPROPERTIES (
'event_time’=’order_time’,
'minute_granualrity’=’1’) AS
SELECT order_time, country, sex, sum(quantity), max(quantity),
count(user_id), sum(price),avg(price)
FROM sales GROUP BY order_time, country, sex
```

c) 用户可不用创建所有时间粒度的聚合表，系统支持自动roll-up上卷，如：已创建了Day粒度的聚合表，当查询Year、Month粒度的group by聚合时，系统会基于已聚合好的Day粒度值推算出Year、Month粒度的聚合值：

```
//创建天粒度聚合表
CREATE DATAMAP agg_day
ON TABLE sales USING "timeseries"
DMPROPERTIES (
'event_time’=’order_time’,
'day_granualrity’=’1’) AS
SELECT order_time, country, sex, sum(quantity), max(quantity),
count(user_id), sum(price),avg(price)
FROM sales GROUP BY order_time, country, sex
```

（Year、Month粒度的聚合查询，可用上面创建的agg_day上卷）

```
SELECT timeseries(order_time, ‘month’), sum(quantity)
FROM sales group by timeseries(order_time,’month’)
SELECT timeseries(order_time, ‘year’), sum(quantity)
FROM sales group by timeseries(order_time, ’year’)
```

此特性为Alpha特性，当前时间粒度支持设置为1，比如：支持按1天聚合，暂不支持指定3天，5天的粒度进行聚合，下个版本将支持。支持自动上卷（Year,Month,Day,Hour,Minute）

流式准实时入库，提升数据时效性同时避免小文件问题

在许多业务场景中，数据时效性是一个非常重要的指标，即数据产生多久后可被查询。而在现有的开源大数据方案中，能做实时入库和实时查询的方案非常少，而且实时入库常常伴随着小文件的产生，所以一般都需要多个存储系统组合使用，如HBase做实时入库，每天导出为Parquet文件后用Impala做分析，数据时效性延迟了一天，而且需要管理多个集群，维护非常困难。

在CarbonData1.3.0中，通过行列混合存储并与Spark的Structured Streaming集成，支持用户准实时导入数据到CarbonData表，并立即可查询这些数据。因为实时导入的表格和历史表是同一张表，所以也不需要在多个表格间来回切换，只需对一张表进行查询即可同时查询实时数据和历史数据。

a) 实时获取数据：

```
val readSocketDF= spark.readStream
.format("socket")
.option("host", "localhost")
.option("port", 9099)
.load()
```

b) 写数据到CarbonData表

```
qry = readSocketDF.writeStream
.format("carbondata")
.trigger(ProcessingTime("5 seconds"))
.option("checkpointLocation", tablePath.getStreamingCheckpointDir)
.option("dbName", "default")
.option("tableName", "carbon_table")
.start()
```

（具体可参考例子/apache/carbondata/examples/CarbonStructuredStreamingExample.scala）

支持标准Hive分区，兼容Spark、Hive的分区使用方法

Hive静态分区和动态分区经过多年的应用，已经为大多数开发人员熟悉，CarbonData 1.3.0中支持标准Hive分区，可以使大数据应用更容易地迁移到CarbonData方案上来。

因为CarbonData具备索引能力，用户在建立分区的同时，可以选择与SORT_COLUMNS组合使用，在分区数据里建立MDK索引，起到多级索引的效果，满足任意维度组合的快速过滤查询，做到一份数据满足多种应用场景

例如创建下面表，设置productDate作为partition字段，数据按天进行分区；再通过SORT_COLUMNS建立多维索引。这样即可按照productDate，productName,
storeProvince,storeCity任意过滤组合快速查询数据。

```
CREATE TABLE IF NOT EXISTS productSalesTable (
productName STRING,
storeProvince STRING,
storeCity STRING,
saleQuantity INT,
revenue INT)
PARTITIONED BY (productDate DATE)
STORED BY 'carbondata'
TBLPROPERTIES('SORT_COLUMNS' = 'productName, storeProvince, storeCity')
```

支持CREATE TABLE AS SELECT语法

CTAS (CREATETABLE AS
SELECT)允许用户从Parquet/Hive/CarbonData表中SELECT来创建新的CarbonData表，
方便用户把Parquet、 Hive格式数据转成CarbonData格式数据。

```
CREATE TABLE carbon_table STORED BY 'carbondata' AS SELECT * FROM parquet_table
```

支持指定导入批次进行查询

CarbonData每批次导入的数据，会产生一个segment，在1.3.0里用户可以指定Segment
ID对某个Segment的数据进行查询，即：用户可以指定数据批次查询。

a) 查询Segment ID列表

```
SHOW SEGMENTS FOR TABLE <databasename>.<table_name>
```

b) 设置Segment ID

```
SET carbon.input.segments.<databasename>.<table_name> = <list of segment IDs>
```

(具体可参考例子：/apache/carbondata/examples/QuerySegmentExample.scala)

Apache CarbonData官网：carbondata.apache.org。1.3.0下载地址：https://dist.apache.org/repos/dist/release/carbondata/1.3.0