Spark 2.1.0与CarbonData 1.0.0集群模式部署及使用入门指南

本文作者：李寅威，从事大数据、机器学习方面的工作，目前就职于CVTE
联系方式：微信（coridc），邮箱（251469031@qq.com）
原文链接：Spark2.1.0 + CarbonData1.0.0集群模式部署及使用入门

1 引言

Apache CarbonData是一个面向大数据平台的基于索引的列式数据格式，由华为大数据团队贡献给Apache社区，目前最新版本是1.0.0版。介于目前主流大数据组件应用场景的局限性，CarbonData诞生的初衷是希望通过仅保存一份数据来满足不同的应用场景，如：

- OLAP
- 顺序存取（Sequential Access）
- 随机存取（Random Access）

CarbonData也被评为2016年的BLACKDUCK奖，有关CarbonData的相关资料如下：

- 官网地址： http://carbondata.apache.org
- Github： https://github.com/carbondata/carbondata
- Mailing list： dev@carbondata.incubator.apache.org
- cwiki：https://cwiki.apache.org/confluence/display/CARBONDATA/CarbonData+Home
- Jira地址： https://issues.apache.org/jira/browse/CARBONDATA

本文主要介绍Spark2.1.0 + CarbonData1.0.0集群模式部署流程，并辅以一个小案例来讲解如何在Spark shell下使用CarbonData。

2 准备工作

2.1 集群规划

id	hostname	mem	cpu	storage
1	master	32G	Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz	SATA3 7200RPM 4T
2	slave1	32G	Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz	SATA3 7200RPM 8T

id	hostname	mem	cpu	storage
3	slave2	32G	Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz	SATA3 7200RPM 8T
4	slave3	32G	Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz	SATA3 7200RPM 8T
5	slave4	32G	Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz	SATA3 7200RPM 8T

2.2 系统环境

操作系统

下载地址：http://mirrors.163.com/
建议版本：Unix-like environment (Linux, Mac OS X)
版本查看：

```
# 示例（CentOS）
[hadoop@master ~]$ cat /etc/redhat-release
CentOS release 6.8 (Final)
```

JDK

下载地址：http://www.oracle.com/technetwork/java/javase/downloads/index.html
建议版本：JDK1.8.0+
版本查看：

```
[hadoop@master ~]$ java -version
java version "1.8.0_60"
```

Git

下载地址：https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
建议版本：无
版本查看：

```
[hadoop@master ~]$ git --version
git version 1.7.1
```

Maven

下载地址：https://maven.apache.org/download.cgi
建议版本：3.0.4
版本查看：

```
[hadoop@master ~]$ mvn -v
Apache Maven 3.0.4 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-11T00:41:47+0
8:00)
Maven home: /opt/maven-3.0.4
```

Hadoop

下载地址：http://hadoop.apache.org/#Download+Hadoop
建议版本：2.7.2
版本查看：

```
[hadoop@master ~]$ hadoop version
Hadoop 2.7.2

[hadoop@master ~]$ echo $HADOOP_HOME
/opt/hadoop-2.7.2
```

Scala

下载地址：http://www.scala-lang.org/
建议版本：2.11.x
版本查看：

```
[hadoop@master ~]$ scala -version
Scala code runner version 2.11.8 -- Copyright 2002-2016, LAMP/EPFL
```

Spark

下载地址：http://spark.apache.org/downloads.html
建议版本：2.1.0
部署模式：Standalone/YARN
版本查看：

```
[hadoop@master spark-2.1.0]$ ./bin/spark-submit --version
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/

Using Scala version 2.11.8, Java HotSpot(TM) 64-Bit Server VM, 1.8.0_60

[hadoop@master ~]$ echo $SPARK_HOME
/opt/spark-2.1.0
```

Thrift

下载地址：http://thrift.apache.org/download
建议版本：0.9.3
版本查看：

```
[hadoop@master ~]$ thrift -version
Thrift version 0.9.3
```

3 编译及部署

3.1 编译

Step 1：源码下载

```
$ git clone https://github.com/apache/incubator-carbondata.git carbondata
```

Step 2：修改Maven私有仓库地址（可选）

由于网络原因，从Maven中央仓库下载jar包可能非常慢，大家可根据自己的实际情况修改为企业内部私有仓库或阿里云等外部源，如：

```
# 修改conf/setting.xml文件
<mirrors>
<mirror>
  <id>nexus</id>
  <name>nexus</name>
  <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
  <mirrorOf>*</mirrorOf>
</mirror>
</mirrors>
```

Step 3：编译打包

```
[hadoop@master ~]$ cd carbondata
[hadoop@master carbondata]$ mvn clean package -DskipTests -Pspark-2.1 -Dspark.version=2.1.0 -Phadoop-2.7.2
```

在编译打包的过程中，maven会自动下载所依赖的jar包，但可能还会有部分jar包无法下载成功导致打包失败的情况，此时需要我们手动去网上下载并将对应的jar包放到Maven localRepository的对应目录下并重新执行上述命令，执行成功后，会出现以下提示：

```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] Apache CarbonData :: Parent ..... SUCCESS [  1.319 s]
[INFO] Apache CarbonData :: Common ..... SUCCESS [16:82 min]
[INFO] Apache CarbonData :: Core ..... SUCCESS [03:23 min]
[INFO] Apache CarbonData :: Processing ..... SUCCESS [  8.623 s]
[INFO] Apache CarbonData :: Hadoop ..... SUCCESS [  6.237 s]
[INFO] Apache CarbonData :: Spark Common ..... SUCCESS [ 52.524 s]
[INFO] Apache CarbonData :: Spark2 ..... SUCCESS [ 50.118 s]
[INFO] Apache CarbonData :: Spark Common Test ..... SUCCESS [ 25.072 s]
[INFO] Apache CarbonData :: Assembly ..... SUCCESS [  5.521 s]
[INFO] Apache CarbonData :: Spark2 Examples ..... SUCCESS [  8.742 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

```
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 29:42 min
[INFO] Finished at: 2017-03-11T09:26:38+08:00
[INFO] Final Memory: 211M/2513M
[INFO] ------------------------------------------------------------------------
```

3.2 集群模式部署

注：本节是基于Spark Standalone模式的配置部署，Spark on Yarn模式下的配置部署类似但不完全相同，具体可参照文档：
- Installing and Configuring CarbonData on Standalone Spark Cluster
- Installing and Configuring CarbonData on “Spark on YARN” Cluster

部署步骤如下：

Step 1：复制CarbonData jar包

```
# 创建目录并拷贝
[hadoop@master spark-2.1.0]$ mkdir $SPARK_HOME/carbonlib
[hadoop@master spark-2.1.0]$ cp -r ~/carbondata/assembly/target/scala-2.11/carbondata_2.11-1.0.0-incubating-shade-hadoop2.7.2.jar ./carbonlib

# 编辑环境变量
[hadoop@master ~]$ vi /opt/spark-2.1.0/conf/spark-env.sh

# 添加以下配置
export SPARK_CLASSPATH=$SPARK_CLASSPATH:${SPARK_HOME}/carbonlib/*
```

Step 2：配置carbon.properties

```
# 拷贝至spark conf目录
[hadoop@master ~]$ cp ~/carbondata/conf/carbon.properties.template /opt/spark-2.1.0/conf/carbon.properties

# 修改配置
#Mandatory. Carbon Store path
carbon.storelocation=hdfs://master:9000/opt/carbonStore

#Base directory for Data files
carbon.ddl.base.hdfs.url=hdfs://master:9000/opt/data
```

```
#lock文件存储方式
carbon.lock.type=HDFSLOCK
```

Step 3：配置spark-defaults.conf

```
[hadoop@master ~]$ vi /opt/spark-2.1.0/conf/spark-defaults.conf

# 添加以下配置
spark.executor.extraJavaOptions       -Dcarbon.properties.filepath=/opt/spark-2.1.0/conf/carbon.properties
spark.driver.extraJavaOptions         -Dcarbon.properties.filepath=/opt/spark-2.1.0/conf/carbon.properties
```

Step 4：分发至集群各节点

```
# Spark - conf
[hadoop@master ~]$ scp -r /opt/spark-2.1.0/conf/ hadoop@slave1:/opt/spark-2.1.0/
[hadoop@master ~]$ scp -r /opt/spark-2.1.0/conf/ hadoop@slave2:/opt/spark-2.1.0/
[hadoop@master ~]$ scp -r /opt/spark-2.1.0/conf/ hadoop@slave3:/opt/spark-2.1.0/
[hadoop@master ~]$ scp -r /opt/spark-2.1.0/conf/ hadoop@slave4:/opt/spark-2.1.0/

# Spark - carbonlib
[hadoop@master ~]$ scp -r /opt/spark-2.1.0/carbonlib/ hadoop@slave1:/opt/spark-2.1.0/
[hadoop@master ~]$ scp -r /opt/spark-2.1.0/carbonlib/ hadoop@slave2:/opt/spark-2.1.0/
[hadoop@master ~]$ scp -r /opt/spark-2.1.0/carbonlib/ hadoop@slave3:/opt/spark-2.1.0/
[hadoop@master ~]$ scp -r /opt/spark-2.1.0/carbonlib/ hadoop@slave4:/opt/spark-2.1.0/
```

4 数据准备

4.1 数据描述

为更聚焦于整个流程，我们的数据集仅含一张表，共21个字段，详细信息如下：

ID	COLUMN	TYPE	CARDINALITY	COMMENT
1	id	BIGINT	TOTAL_NUM	ID
2	order_code	STRING	TOTAL_NUM	订单编号
3	sales_area_id	INT	100	销售区域ID
4	sales_id	INT	10000	销售人员ID
5	order_inputer	INT	100	录单人员ID
6	pro_type	STRING	1000	产品型号
7	currency	INT	50	订单币种
8	exchange_rate	DECIMAL	1000	当前汇率
9	unit_cost_price	DECIMAL	10000	成本单价
10	unit_selling_price	DECIMAL	10000	销售单价
11	order_num	INTEGER	1000	订单数量
12	order_amount	DECIMAL	/	订单金额
13	order_discount	DOUBLE	9	订单折扣
14	order_account_amount	DECIMAL	/	实际金额
15	order_time	TIMESTAMP	8000000	下单时间
16	delivery_channel	INT	80	发货渠道ID
17	delivery_address	STRING	10000000	发货地址
18	recipients	STRING	10000000	收件人
19	contact	STRING	10000000	联系方式
20	delivery_date	DATE	10000	发货日期
21	comments	STRING	10000000	备注

4.2 数据生成

我们编写一段程序来解决数据批量生成的问题，为方便有兴趣的童鞋直接使用，所有代码都写在一个Java类中，相关配置（如数据量等）可直接在代码中修改。当然大家也完全可以用自己的数据进行测试。数据生成的代码如下（注：需使用JDK1.8及以上版本编译运行）：

```
package com.cvte.bigdata.pt;
```

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.UUID;
import java.util.concurrent.ArrayBlockingQueue;

public class BatchDataGenerate implements Runnable {

    // total record
    private final Integer TOTAL_NUM = 1 * 10000 * 10000;
    // batch size of writing to file and echo infos to console
    private final Integer BATCH_SIZE = 10 * 10000;
    // capacity of the array blocking queue
    private final Integer QUEUE_CAPACITY = 100 * 10000;
    // csv file path that save the data
    private final String DEST_FILE_PATH = "E:₩₩order_detail.csv";
    // the queue which the write thread write into and the read thread read from
    private ArrayBlockingQueue<String> queue = new ArrayBlockingQueue<>(QUEUE_CAPACITY
);

    // the cardinalities of the fields(dimensions) which means the distinct
    // number of a column
    private final int CARDINALITY_SALES_AREA_ID = 100;
    private final int CARDINALITY_SALES_ID = 10000;
    private final int CARDINALITY_ORDER_INPUTER = 100;
    private final int CARDINALITY_PRO_TYPE = 1000;
    private final int CARDINALITY_CURRENCY = 50;
    private final int CARDINALITY_EXCHANGE_RATE = 1000;
    private final int CARDINALITY_UNIT_COST_PRICE = 10000;
    private final int CARDINALITY_UNIT_SELLING_PRICE = 10000;
    private final int CARDINALITY_ORDER_NUM = 1000;
    private final int CARDINALITY_ORDER_DISCOUNT = 9;
    private final int CARDINALITY_ORDER_TIME = 8000000;
    private final int CARDINALITY_DELIVERY_CHANNEL = 80;
    private final int CARDINALITY_DELIVERY_ADDRESS = 10000000;
    private final int CARDINALITY_RECIPIENTS = 10000000;
    private final int CARDINALITY_CONTACT = 10000000;
    private final int CARDINALITY_DELIVERY_DATE = 10000;
    private final int CARDINALITY_COMMENTS = 10000000;

    @Override
```

```
public void run() {
    try {
        if ("tGenerate".equals(Thread.currentThread().getName())) {
            // data generating thread
            generateData();
        } else if ("tWrite".equals(Thread.currentThread().getName())) {
            // data writing thread
            saveDataToFile();
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
* @Description: generating data for table order_details and called by thread tGenerate
* @param @throws InterruptedException
* @return void
* @throws
* @Author liyinwei
* @date 2017年3月9日 下午7:19:16
 */
private void generateData() throws InterruptedException {
    for (int i = 0; i < TOTAL_NUM; i++) {
        StringBuffer sb = new StringBuffer();
        sb.append(i + 1)
                // 2.订单编号
                .append(",").append(UUID.randomUUID())
                // 3.销售区域ID
                .append(",").append((i+3) % CARDINALITY_SALES_AREA_ID)
                // 4.销售人员ID
                .append(",").append((i + 4) % CARDINALITY_SALES_ID)
                // 5.录单人员ID
                .append(",").append((i + 5) % CARDINALITY_ORDER_INPUTER)
                // 6.产品型号
                .append(",").append("PRO_TYPE_" + (i + 6) % CARDINALITY_PRO_TYPE)
                // 7.订单币种
                .append(",").append((i + 7) % CARDINALITY_CURRENCY)
                // 8.当前汇率
                .append(",").append((i + 8) % CARDINALITY_EXCHANGE_RATE)
                // 9.成本单价
                .append(",").append((i + 9) % CARDINALITY_UNIT_COST_PRICE)
                // 10.销售单价
                .append(",").append((i + 10) % CARDINALITY_UNIT_SELLING_PRICE)
```

```
                // 11.订单数量
                .append(",").append((i + 11) % CARDINALITY_ORDER_NUM)
                // 12.订单金额
                .append(",").append(((i + 10) % CARDINALITY_UNIT_SELLING_PRICE) * ((i + 11) % CA
RDINALITY_ORDER_NUM))
                // 13.订单折扣
                .append(",").append(String.format("%.2f", (i + 13) % CARDINALITY_ORDER_DISCOU
NT * 0.1))
                // 14.实际金额
                .append(",")
                .append(String.format("%.2f",
                        ((i + 10) % CARDINALITY_UNIT_SELLING_PRICE) * ((i + 11) % CARDINALITY_OR
DER_NUM)
                                * ((i + 13) % CARDINALITY_ORDER_DISCOUNT * 0.1)))
                // 15.下单时间
                .append(",")
                .append(LocalDateTime.now().plusSeconds(i % CARDINALITY_ORDER_TIME)
                        .format(DateTimeFormatter.ofPattern("yyyy-MM-dd hh:mm:ss")))
                // 16.发货渠道ID
                .append(",").append((i + 16) % CARDINALITY_DELIVERY_CHANNEL)
                // 17.发货地址
                .append(",").append("DELIVERY_ADDRESS_" + (i + 17) % CARDINALITY_DELIVERY_AD
DRESS)
                // 18.收件人
                .append(",").append("RECIPIENTS_" + (i + 18) % CARDINALITY_RECIPIENTS)
                // 19.联系方式
                .append(",").append(13800000000l + (i + 19) % CARDINALITY_CONTACT)
                // 20.发货日期
                .append(",").append(LocalDate.now().plusDays(i % CARDINALITY_DELIVERY_DATE))
                // 21.备注
                .append(",").append("RECIPIENTS_" + (i + 21) % CARDINALITY_COMMENTS);

            queue.put(sb.toString());
            if (i % BATCH_SIZE == 0) {
                System.out.println(i + " records have generated successfully.");
                System.out.println("current queue length is: " + queue.size());
            }
        }
    }

    /**
    * @Description: writing data from array block queue to file and called by thread tWrite
    * @param @throws InterruptedException
    * @param @throws IOException
    * @return void
    * @throws
```

```
 * @Author liyinwei
 * @date 2017年3月9日 下午8:16:59
 */
private void saveDataToFile() throws InterruptedException, IOException {
    int i = 0;
    StringBuffer sb = new StringBuffer();
    while (true) {
        sb.append(queue.take()).append("₩n");
        i++;
        if (i % BATCH_SIZE == 0) {
            Files.write(Paths.get(DEST_FILE_PATH), sb.toString().getBytes(),
                    StandardOpenOption.CREATE, StandardOpenOption.APPEND);
            sb.setLength(0);

            System.out.println(i + " records have written to file successfully.");
            System.out.println("current queue length is: " + queue.size());
        }
    }
}

public static void main(String[] args) {
    BatchDataGenerate batchDataGenerate = new BatchDataGenerate();

    // data generating thread
    Thread thread1 = new Thread(batchDataGenerate, "tGenerate");
    // data writing thread
    Thread thread2 = new Thread(batchDataGenerate, "tWrite");

    thread1.start();
    thread2.start();
}
}
```

以上代码的逻辑比较简单，构造两个线程，一个用于生成数据并保存在队列中，另一个用于从队列中取数据并保存在文件中。代码中几个关键的变量（参数）介绍如下：

- TOTAL_NUM：需要生成的数据总条数
- BATCH_SIZE：控制台打印进度信息及队列当前信息的数据条数间隔，同时也是每批次写文件的数据量大小
- QUEUE_CAPACITY：队列大小，若队列满则阻塞写线程
- DEST_FILE_PATH：数据文件存储目录
- queue：缓存队列

运行上述代码，稍等片刻，我们就可以在目标路径中找到order_detail.csv文件，3亿条数据占用61.8G存储空间。

4.3 数据上传

接着，我们将order_detail.csv上传至HDFS中：

```
[hadoop@master ~]$ hadoop fs -put /home/hadoop/order_detail.csv /data/carbondata/pt
```

5 启动spark-shell

在使用CarbonData前，有以下先决条件 ：

- Hadoop HDFS及YARN已安装及启动
- Spark已安装并在每个节点启动
- CarbonData用户拥有读写HDFS的权限

5.1 简介

CarbonData对Spark已有较好支持，我们可将carbondata的jar包传入spark-shell来使用CarbonData：

```
./bin/spark-shell --jars <carbondata assembly jar path>
```

进入上述Shell后，会自动生成一个名为spark的SparkSession及一个名为sc的SparkContext。

5.2 案例

样例代码如下：

```
[hadoop@master ~]$ cd /opt/spark-2.1.0
[hadoop@master spark-2.1.0]$ carbondata_jar=./carbonlib/$(ls -1 carbonlib |grep "^carbondata_.*₩.jar$")
[hadoop@master spark-2.1.0]$ ./bin/spark-shell --num-executors 4 --executor-cores 1 --driver-memory 5g --total-executor-cores 16 --executor-
memory 5g --master spark://master:7077 --jars ${carbondata_jar}
```

细心的小伙伴可能已经发现，在样例代码中，我们在启动spark shell时额外添加了几个参数，这里要强调一点的是，Spark的配置及spark shell的配置会很大程度影响查询性能，因此，加入我们需要做组件性能的横向对比，需注意以下事项：
1. 保持参数的一致性，强调单一变量原则；
2. 需为集群配置合理的参数，以尽可能地发挥组件的性能；

关于Spark的配置可参照以下文档：
- Cluster Launch Scripts - Spark Standalone Mode
- Launching Applications with spark-submit - Submitting Applications

6 创建CarbonSession

6.1 简介

接下来我们需要创建一个CarbonSession来对CarbonData相关的内容进行操作：

```
# import packages
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.CarbonSession._

# create CarbonSession
val carbon = SparkSession.builder().config(sc.getConf).getOrCreateCarbonSession("<hdfs store path>")
```

6.2 案例

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.CarbonSession._
val carbon = SparkSession.builder().config(sc.getConf).getOrCreateCarbonSession("hdfs://master:9000/opt/carbonStore")
```

至此，就完成了CarbonSession的创建。

7 创建数据库及数据表

7.1 简介

关于CarbonData表的创建，可参照CarbonData官方文档中的CREATE TABLE部分，为充分发挥CarbonData的优势，我们需注意以下内容：

- 字典编码（Dictionary Encoding）

默认情况下CarbonData会自动为String类型的字段创建字典，而非String类型的字段不会创建字典。我们可根据实际需求进行调整，如取消对部分高基列的字典编码等，以下为对应的语法：

TBLPROPERTIES ("DICTIONARY_EXCLUDE"="column1, column2") TBLPROPERTIES ("DICTIONARY_INCLUDE"="column1, column2")

- 列组（Column group）

对于经常一起出现的列，我们可以将其设置为列组，被设置为列组的列会按行的方式进行存储，然后与其它列或列组一起按列存储。默认情况下，CarbonData不自动对列进行组合。以下为对应的语法：

TBLPROPERTIES ("COLUMN_GROUPS"="(column1, column3), (Column4,Column5,Column6)")

块大小（Table Block Size）

CarbonData支持1M-2018M大小的块，默认为1024M，我们可按需进行调整，以下为对应的语法：

TBLPROPERTIES ("TABLE_BLOCKSIZE"="512 MB")

倒排索引（Inverted Index）

倒排索引在压缩率和查询效率方面有非常大的作用，这对低基维度（默认情况下，CarbonData会将所有非数值类型的字段视为维度，而将数值型字段视为度量值）效果尤其明显。默认情况下，CarbonData会自动创建，我们可根据实际需求进行调整，如取消对部分高级维度的倒排索引。以下为对应的语法：

TBLPROPERTIES ("NO_INVERTED_INDEX"="column1, column3")

7.2 案例

在本文的例子中，我们可设计建表语句如下：

```
CREATE TABLE
IF NOT EXISTS pt.order_detail (
    id BIGINT,
    order_code STRING,
    sales_area_id INT,
    sales_id INT,
    order_inputer INT,
    pro_type STRING,
    currency INT,
    exchange_rate DECIMAL,
    unit_cost_price DECIMAL,
    unit_selling_price DECIMAL,
    order_num INTEGER,
    order_amount DECIMAL,
    order_discount DOUBLE,
    order_account_amount DECIMAL,
    order_time TIMESTAMP,
    delivery_channel INT,
    delivery_address STRING,
    recipients STRING,
    contact STRING,
    delivery_date DATE,
    comments STRING
) STORED BY 'carbondata' TBLPROPERTIES (
    'COLUMN_GROUPS' = '(recipients,contact)',
    'DICTIONARY_EXCLUDE' = 'comments',
    'DICTIONARY_INCLUDE' = 'sales_area_id,sales_id',
    'NO_INVERTED_INDEX' = 'id,order_code'
)
```

在spark-shell中，我们可通过以下命令建表：

```
carbon.sql("CREATE TABLE IF NOT EXISTS pt.order_detail ( id BIGINT, order_code STRING, sales_area_id INT, sales_id INT, order_inputer INT, pro_type STRING, currency INT, exchange_rate DECIMAL, unit_cost_price DECIMAL, unit_selling_price DECIMAL, order_num INTEGER, order_amount DECIMAL, order_discount DOUBLE, order_account_amount DECIMAL, order_time TIMESTAMP, delivery_channel INT, delivery_address STRING, recipients STRING, contact STRING, delivery_date DATE, comments STRING ) STORED BY 'carbondata' TBLPROPERTIES ( 'COLUMN_GROUPS' = '(recipients,contact)', 'DICTIONARY_EXCLUDE' = 'comments', 'DICTIONARY_INCLUDE' = 'sales_area_id,sales_id', 'NO_INVERTED_INDEX' = 'id,order_code' )")
```

8 数据查询

8.1 数据导入

CarbonData支持两种方式的数据导入，分别为：
- 直接通过CSV文件导入CarbonData表
- 通过spark-sql API导入

8.1.1 导入方式一

8.1.1.1 简介

第一种数据导入方式是将原始数据从CSV文件导入CarbonData表形成Carbon格式的文件，其语法为：

LOAD DATA [LOCAL] INPATH 'folder_path' INTO TABLE [db_name.]table_name OPTIONS(property_name=property_value, ...)

由上述命令中我们可以看到，CarbonData支持从LOCAL及非LOCAL（如HDFS）位置导入，OPTIONS选项支持的内容可查看DML Operations on CarbonData。

需要注意的是，CarbonData并不会自动根据顺序识别CSV文件中的列与数据表字段间的关系，为此有两种解决方案，

第一种是在CSV文件的第一行加上列名，与数据保持相同的分隔符，如：

id,username,age
1,zhangsan,20
2,lisi,21

第二种是在load data命令中添加fileheader，如：

OPTIONS('FILEHEADER'='column1,column2')

8.1.1.2 案例

在spark-shell中，我们可以用benchmark{}来进行性能测试，使用benchmark{}需导入包：

```
import org.apache.spark.sql.catalyst.util._
```

结合benchmark，我们一起来看一下以下的案例：

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.CarbonSession._
import org.apache.spark.sql.catalyst.util._

val carbon = SparkSession.builder().config(sc.getConf).getOrCreateCarbonSession("hdfs://master:9000/opt/carbonStore")

val src="hdfs://master:9000/data/carbondata"

benchmark{carbon.sql(s"load data inpath '$src/order_detail.csv' into table pt.order_detail OPTIONS('DELIMITER'=',','fileheader'='id,order_code,sales_area_id,sales_id,order_inputer,pro_type,currency,exchange_rate,unit_cost_price,unit_selling_price,order_num,order_amount,order_discount,order_account_amount,order_time,delivery_channel,delivery_address,recipients,contact,delivery_date,comments')")}
```

8.1.2 导入方式二

8.1.2.1 简介

CarbonData支持将DataFrame转换成一个Carbon File，详情可参照文档：Carbondata Interfaces。

8.1.2.2 案例

```
// User can create a DataFrame from any data source or transformation.
val df = ...

// Write data
// User can write a DataFrame to a carbon file
 df.write
   .format("org.apache.spark.sql.CarbonSource")
   .option("tableName", "carbontable")
   .mode(SaveMode.Overwrite)
   .save()
```

```
// read carbon data by data source API
df = carbonContext.read
  .format("org.apache.spark.sql.CarbonSource")
  .option("tableName", "carbontable")
  .load("/path")
```

8.1.3 badrecord处理

8.1.3.1 简介

CarbonData Data Load发生后有两种状态，一种是成功，另一种则是部分成功。其中部分成功是由于数据导入过程发现了非法记录（如一条记录的列数与表的列数不匹配，官方称为badrecord），badrecord存储在carbon.badrecords.location所配置的目录下。

8.1.3.2 案例

案例如下：

```
CarbonProperties.getInstance().addProperty("carbon.badRecords.location","hdfs://master:9000/data/carbondata/badrecords/")

benchmark{carbon.sql(s"load data inpath '$src/order_detail.csv' into table pt.order_detail OPTIONS('DELIMITER'=',','bad_records_logger_enable'='true','fileheader'='id,order_code,sales_area_id,sales_id,order_inputer,pro_type,currency,exchange_rate,unit_cost_price,unit_selling_price,order_num,order_amount,order_discount,order_account_amount,order_time,delivery_channel,delivery_address,recipients,contact,delivery_date,comments')")}
```

8.1.4 数据校验

8.1.4.1 简介

在跑完Load Data作业后，我们需要对数据进行多角度的校验，如数据总量、是否存在格式不匹配（相应字段值为NULL）、是否存在badrecord等。

8.1.4.2 案例

```
carobon.sql("select count(1) from pt.order_detail").show
carbon.sql("select * from pt.order_detail").show
```

8.2 数据查询

CarbonData支持两种方式的数据查询，分别为：
- 通过spark-shell
- 通过spark-sql api

8.2.1 查询方式一

对于通过spark-shell的方式，CarbonData查询的方式很简单，只需在创建好CarbonSession的基础上执行如下语句：

```
carbon.sql("select * from pt.order_detail").show
```

同样的，若要显示性能信息，则添加benchmark{}：

```
benchmark{carbon.sql("select * from pt.order_detail").show}
```

8.2.2 查询方式二

CarbonData还支持通过spark-sql api的方式对表进行查询，如下所示：

```
carbondf.filter($"currency" === "1" and $"contact" === "13800000001").count
```

同理，若要显示性能信息，则添加benchmark{}：

```
benchmark{carbondf.filter($"currency" === "1" and $"contact" === "13800000001").count}
```

附：测试SQL脚本

– OLAP查询

```
select recipients, sum(order_amount) sum_amount from pt.order_detail group by recipients h
aving sum(order_amount) > 20 order by sum_amount desc limit 100
select delivery_date, avg(exchange_rate) from pt.order_detail group by delivery_date having av
g(exchange_rate) > 100 limit 100
```

– 顺序查询

```
select * from pt.order_detail where order_discount < 0.5 and delivery_date > '2018-01-01' limit
10
select recipients, contact from pt.order_detail where order_num < 100 and order_amount < 10
000
```

– 随机查询

```
select * from pt.order_detail where contact='13800000001'
select * from pt.order_detail where sales_area_id=99 and sales_id=9500
```

9 结束语

CarbonData 作为大数据开源软件的BLACKDUCK，它的创新性和影响力已得到认可，随着社区孵化进度的加速，CarbonData在性能及通用性等方面势必有进一步可观的发展，让我们一起拭目以待。