Spark企业级大数据项目实战 第8课

DATAGURU专业数据分析社区

法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

炼数成金逆向收费式网络课程

- Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 http://edu.dataguru.cn

本课内容

- 文件的存储格式
- Spark整合Hive
- 根据日志的实际时间（时间字段）分区
- Kafka的离线API
- kafka数据源清洗入HDFS，保证Exactly-Once
- HDFS原始文件数据源， 清洗入HDFS

1 文件的存储格式-行、列存储

id	name	age
A	狗娃	10
B	狗剩	12
C	狗蛋	13
D	二狗	25
E	三狗	30

- 行式存储：
  - 数据在磁盘中按照行的方式组织和物理存储。
  - 适合对记录的增加、删除、修改
  - 对列的统计分析， 需要消耗大量IO， 因为需要将每列对应的行数据都读取到内存
- 列式存储：
  - 数据在磁盘中按照列的方式组织和物理存储
  - 对记录的增加、删除、修改需要消耗大的IO，效率低
  - 适合统计查询， 一般统计查询针对列进行， 只需要加载需要的列到内存。

行式存储：

A	狗娃	10	B	狗剩	12	C	狗蛋	13	D	二狗	25	E	三狗	30

列式存储：

A	B	C	D	E	狗娃	狗剩	狗蛋	二狗	三狗	10	12	13	25	30

1 文件的存储格式-HDFS文件存储格式

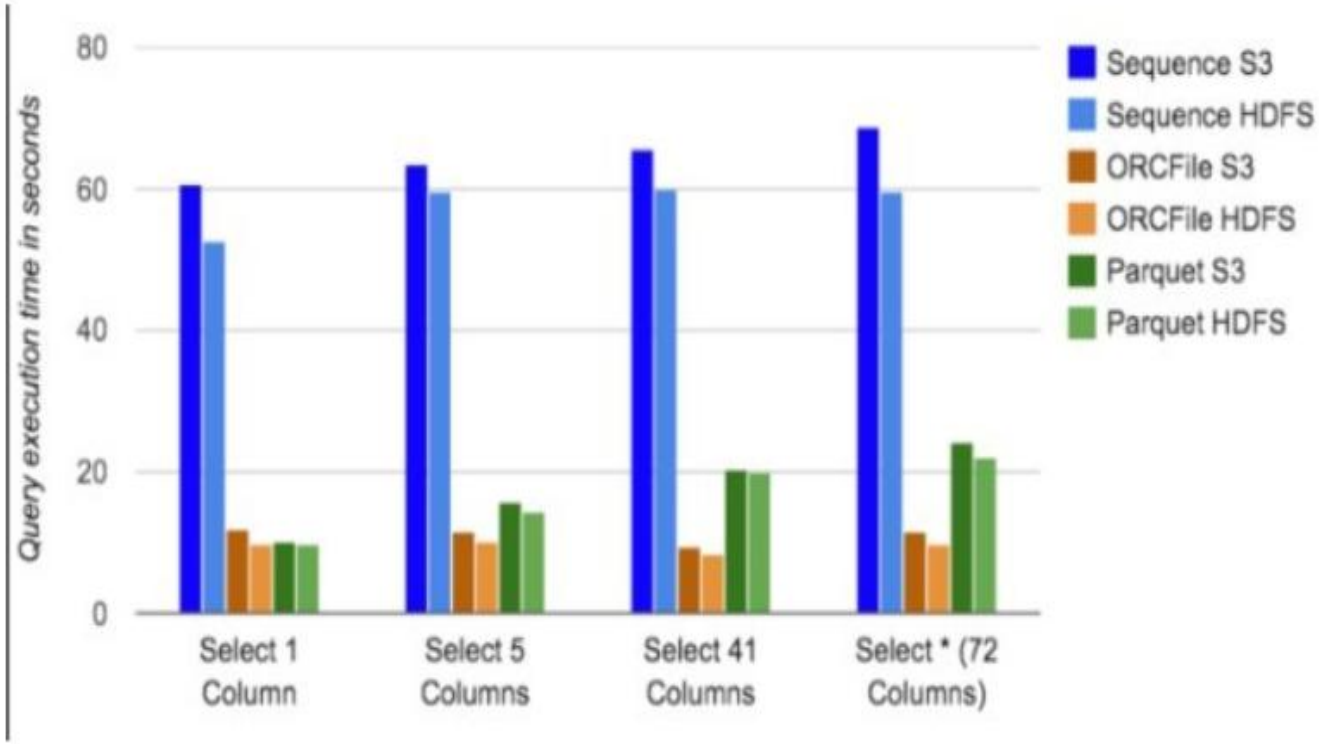
文件格式	类型	存储
TextFile	文本	行存储
SequenceFile	二进制	行存储
RCFile	二进制	列存储
ORC	二进制	列存储
Parquet	二进制	列存储
Avro	二进制	行存储

HDFS的文件格式： Text、Sequence、RCFile、ORC、Parquet等

- **TextFile：** 可读性好， 占用磁盘空间大。
- **SequenceFile：** Hadoop API提供的一种二进制文件， 以<key, value>的形式序列化到文件中。
- **RCFile：** 面向列的数据格式。
- **ORC：** RCFile的升级版， 在压缩编码、查询性能等方面比RCFile做了很多优化。
- **Parquet：** 最初的设计动机是存储嵌套式数据， 如json等。 这类数据存储成列式格式， 实现高效压缩和编码。
- **Avro：** 一种支持数据密集型的二进制文件格式，能够提供更好的序列化和反序列化功能。 Flume就是使用Avro

1 文件的存储格式-HDFS文件存储格式

存储格式	原始大小	编码后大小	压缩后占比
Text	100G	100G	\
RCFile	100G	86G	86%
Parquet	100G	37G	37%
ORC	100G	22G	22%



查询效率

相比其他存储格式， **ORC**在查询性能、存储空间均具有明显优势， 生产上推荐使用**ORC**格式

1 文件的存储格式-如何创建ORC格式文件

指定Hive的文件存储格式

```
create table acclog
(
        srcip string, destip string, proctype string, srcport string, destport string,
        domain string, url string, duration string, acctime string
)  partitioned by (houseid string, dayid string, hourid string)
stored as orc location '/hadoop/kafka/';
```

Spark的DataFrame数据源接口

```
df.write.format("orc").partitionBy("dayid", "hourid").mode(SaveMode.Overwrite).save(outputPath)
```

2 Spark整合Hive

Spark借助Hive的metadata元数据信息， 可以直接操作Hive中的表，
也可以将HDFS上的文件映射为Hive的表， 方便Spark操作数据。

1. 编译Spark二进制包， 需要指定支持Hive

```
./make-distribution.sh --name 2.6.0-cdh5.6.0 --tgz  -Pyarn -Phadoop-2.6
-Phive  -Phive-thriftserver -Dhadoop.version=2.6.0-cdh5.6.0
```

2. 将hive的hive-site.xml复制到spark的conf目录下面

3. 运行Spark程序， 需要指定MySql的JDBC驱动包

```
spark-shell \
  --master yarn \
  --jars /home/hadoop/app/lib/mysql-connector-java-5.1.44-bin.jar
```

2 Spark整合Hive-案例

使用Idea调试
1. 需要将hive-site.xml复制resource目下面。

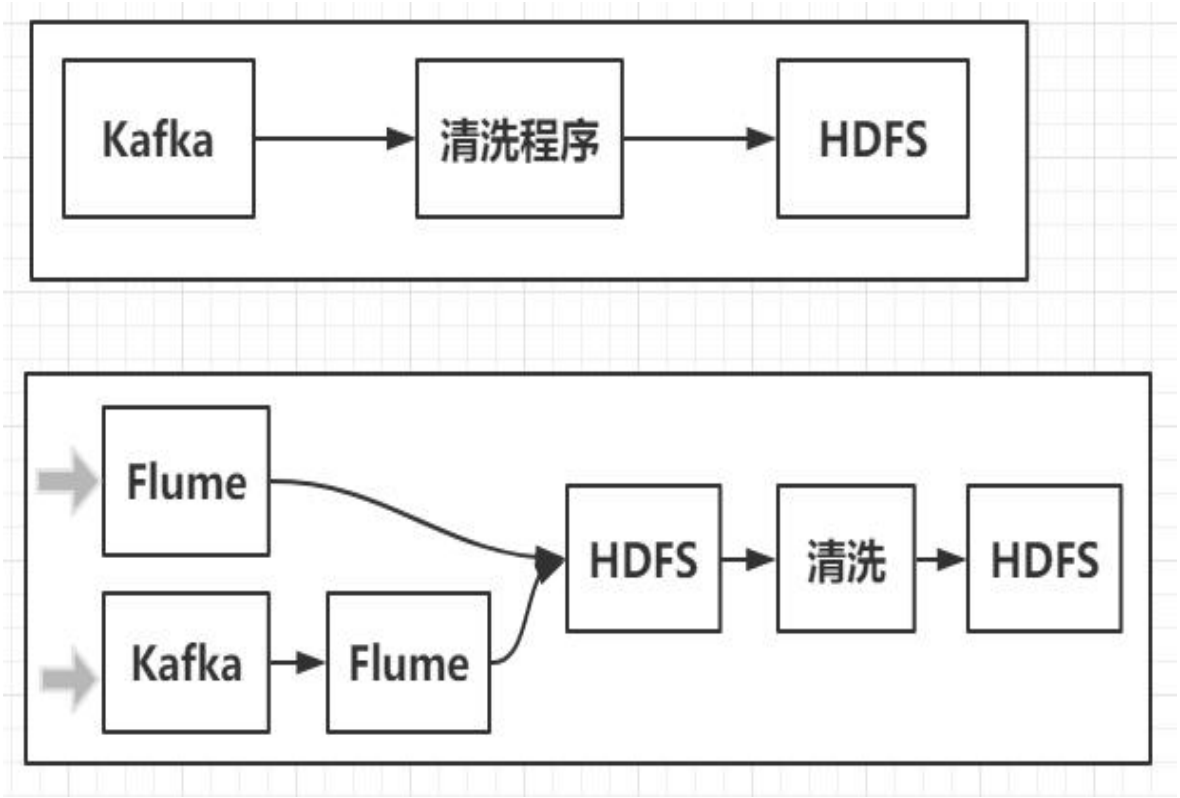
2. 在Pom引入mysql的jdbc依赖

```
// 初始化支持Hive的SQLContext
val sqlContext = new HiveContext(sc)

// 切换hive的数据库test
sqlContext.sql("use test")

// 对Hive中的表统计
val df = sqlContext.sql("select id, count(*) as cnt from tmp group by id ")
df.show()
```

3 根据日志的实际时间（时间字段）分区



场景：

1. 数据源Kafka
   按照每5分钟一个区间统计IP地址的访问次数等
   日志的实际时间（时间字段）在Kafka中不是有序的。
   前后两个offset的日志可能跨越两个5分钟区间

2. 原始数据先入到HDFS
   原始数据入HDFS不能保证是根据日志里面的时间字段分区。

如何解决：

1. 对数据做一次清洗， 将数据按照实际时间划分分区
2. 分析程序对实际时间分区的数据进行分析

3 根据日志的实际时间（时间字段）分区-Hive动态分区

建表语句：
create table testPartition(id int, name string) partitioned by(dayid string) stored as orc;

不使用动态分区插入数据

每次插入数据，需要静态指定具体的分区
insert into testPartition **partition(dayid='20180322')** values(1, 'xiao');

```
# hdfs dfs -ls /user/hive/warehouse/dpi.db/testpartition

hive          0 2018-03-25 00:18 /user/hive/warehouse/dpi.db/testpartition/dayid=20180322
hive          0 2018-03-25 00:20 /user/hive/warehouse/dpi.db/testpartition/dayid=20180323
#
```

使用动态分区

Hive的动态分区默认是关闭的， 需要手工开启：
set hive.exec.dynamic.partition.mode=nonstrict

insert into testPartition **partition (dayid)** values(1, 'xiao', '20180323');

开启动态分区后， 只需要指定分区的字段即可， 分区的值放在最后。

3 根据日志的实际时间（时间字段）分区-Spark的PartitionBy

```
+-----------+----+--------------------+-------------------+--------+------+
|sourceip   |port|url                 |time               |dayid   |hourid|
+-----------+----+--------------------+-------------------+--------+------+
|136.42.33.6 |80  |http://www.baidu.com |2018-03-22 19:50:32|20180322|19    |
|132.92.73.7 |880 |http://www.google.com|2018-03-22 19:30:46|20180322|19    |
|138.52.53.22|68  |http://www.taobao.com|2018-03-22 18:50:25|20180322|18    |
|192.62.93.56|808 |http://www.qq.com    |2018-03-22 18:50:24|20180322|18    |
|101.82.33.78|99  |http://www.baidu.com |2018-03-22 20:50:14|20180322|20    |
|134.72.23.98|123 |http://www.jd.com    |2018-03-22 20:20:31|20180322|20    |
+-----------+----+--------------------+-------------------+--------+------+
```

```
val outputPath = "/tmp/spark"
df.write.format("orc").partitionBy("dayid", "hourid").mode(SaveMode.Overwrite).save(outputPath)
```

根据字段dayid和hourid分区， 注意dayid和hourid必须是df的schema字段

```
[hadoop@spark123 ~]$ hdfs dfs -ls /tmp/spark/dayid=20180322
18/03/25 00:32:27 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platfor
ble
Found 3 items
drwxr-xr-x   - hadoop supergroup          0 2018-03-25 00:26 /tmp/spark/dayid=20180322/hourid=18
drwxr-xr-x   - hadoop supergroup          0 2018-03-25 00:26 /tmp/spark/dayid=20180322/hourid=19
drwxr-xr-x   - hadoop supergroup          0 2018-03-25 00:26 /tmp/spark/dayid=20180322/hourid=20
[hadoop@spark123 ~]$
```

4 Kafka离线API

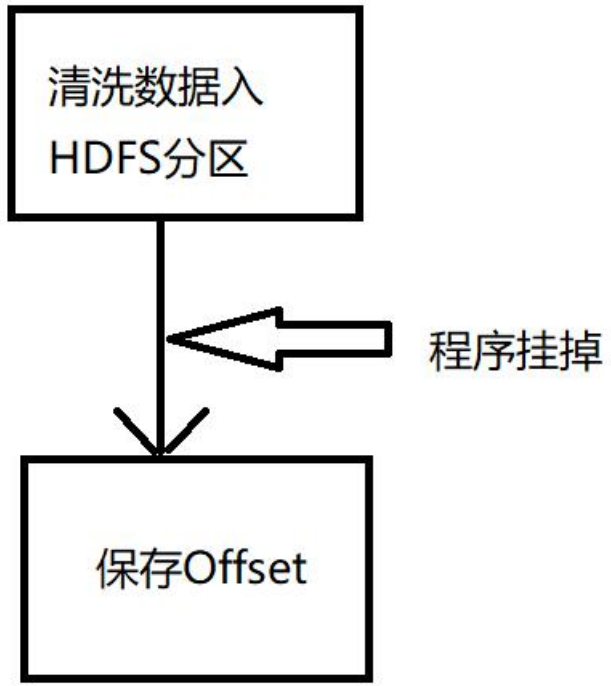
使用Kafka的离线API：KafkaUtils.createRDD

KafkaUtils.createRDD[String, String, StringDecoder, StringDecoder](sc, kafkaParams, offsetRanges)

```
// 每个partition消费最大消息条数
    val maxMsgNumPerPartition = 100000
    val offsetRanges = fromOffsets.keys.map(tp => {
      val fromOffset = fromOffsets(tp)
      val largestOffset = largestOffsets(tp)
      val untilOffset = Math.min(largestOffset, fromOffset + maxMsgNumPerPartition) // 可以限制每次消费的数据量
      OffsetRange(tp, fromOffset, untilOffset)
    }).toArray
```

1. Kafka的离线API可以指定开始和结束的Offset
2. 可以更精确的控制任务需要资源。 可以控制每次消费的最大数据量， 比如控制每个partition消费最大消息条数为10万条（约500M）。
3. 可以方便的实现各种消费语义。

5 kafka数据源清洗入HDFS，保证Exactly-Once



数据总体处理流程： 清洗数据入HDFS分区 -》 保存offset

5 kafka数据源清洗入HDFS，保证Exactly-Once

使用Kafka的离线API清洗数据， 入HDFS

Offset保存在Hbase中

rowkey： topic名称:group名:时间戳

```
hbase(main):101:0* scan 'spark_kafka_offsets'
ROW                                   COLUMN+CELL
 myat:testg:1521880638653             column=offsets:0, timestamp=1521880639140, value=11
 myat:testg:1521880638653             column=offsets:1, timestamp=1521880639140, value=11
 myat:testg:1521880638653             column=offsets:2, timestamp=1521880639140, value=11
 myat:testg:1521880828378             column=offsets:0, timestamp=1521880828717, value=13
 myat:testg:1521880828378             column=offsets:1, timestamp=1521880828717, value=14
 myat:testg:1521880828378             column=offsets:2, timestamp=1521880828717, value=14
 myat:testg:1521880962720             column=offsets:0, timestamp=1521880962861, value=14
 myat:testg:1521880962720             column=offsets:1, timestamp=1521880962861, value=15
 myat:testg:1521880962720             column=offsets:2, timestamp=1521880962861, value=15
```

5 kafka数据源清洗入HDFS，保证Exactly-Once

使用Kafka的离线API清洗数据， 入HDFS

数据处理流程

1. 获取任务运行的BATCH_ID
   如果是第一次运行任务，hbase中没有保存offset， 那么BATCH_ID = TOPIC_NAME + "-" + GROUP_ID + "-1"
   如果不是第一次运行任务，从hbase中获取最新的rowkey， BATCH_ID = BATCH_ID = rowkey.replaceAll(":", "-")

2. 消费Kafka的数据，并根据时间字段分区， 将数据保存到临时目录： outputpath/tmp/BATCH_ID

3. 将临时目录的数据move(rename)到正式分区outputpath/data/, 并将文件名加上前缀BATCH_ID。
   在数据移动到正式分区之前， 删除正式分区下文件名前缀为BATCH_ID的文件， 避免数据重复。

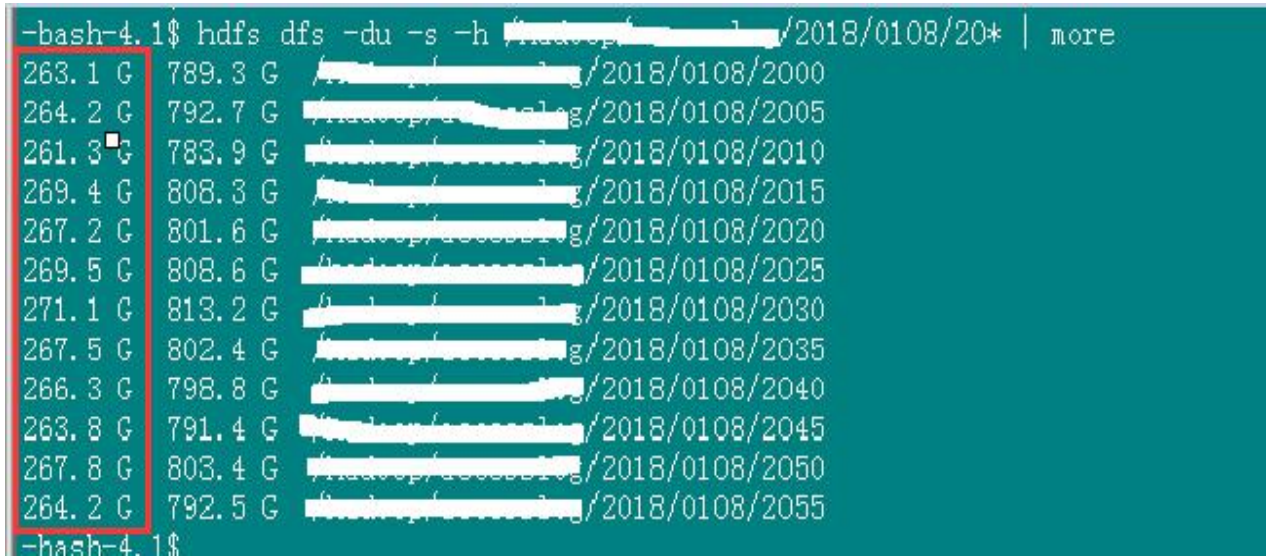
4. 保存offset到hbase
   如果这个批次从kafka消费的数据记录数为0， 不更新offset。
   offset的时间戳为当前系统的最新时间， 精确到毫秒。

6 HDFS原始文件数据源， 清洗入HDFS

使用Kafka的离线API清洗数据， 入HDFS

1. 原始文件按照年/月日/5分钟的目录组织

2. 清洗程序每次读取5分钟的目录，清洗后的数据按照时间字段分区入HDFS分区目录



存在问题？
思考； 如何保证Exactly-Once？

Thanks

FAQ时间