Spark SQL在机票数据处理中的实践

机票-战略发展-大数据-数据挖掘

主讲人：赵法宪

2017.08.21

去哪儿? Qunar.Com 聪明你的旅行

去哪儿? Qunar.Com 聪明你的旅行

目录

01.

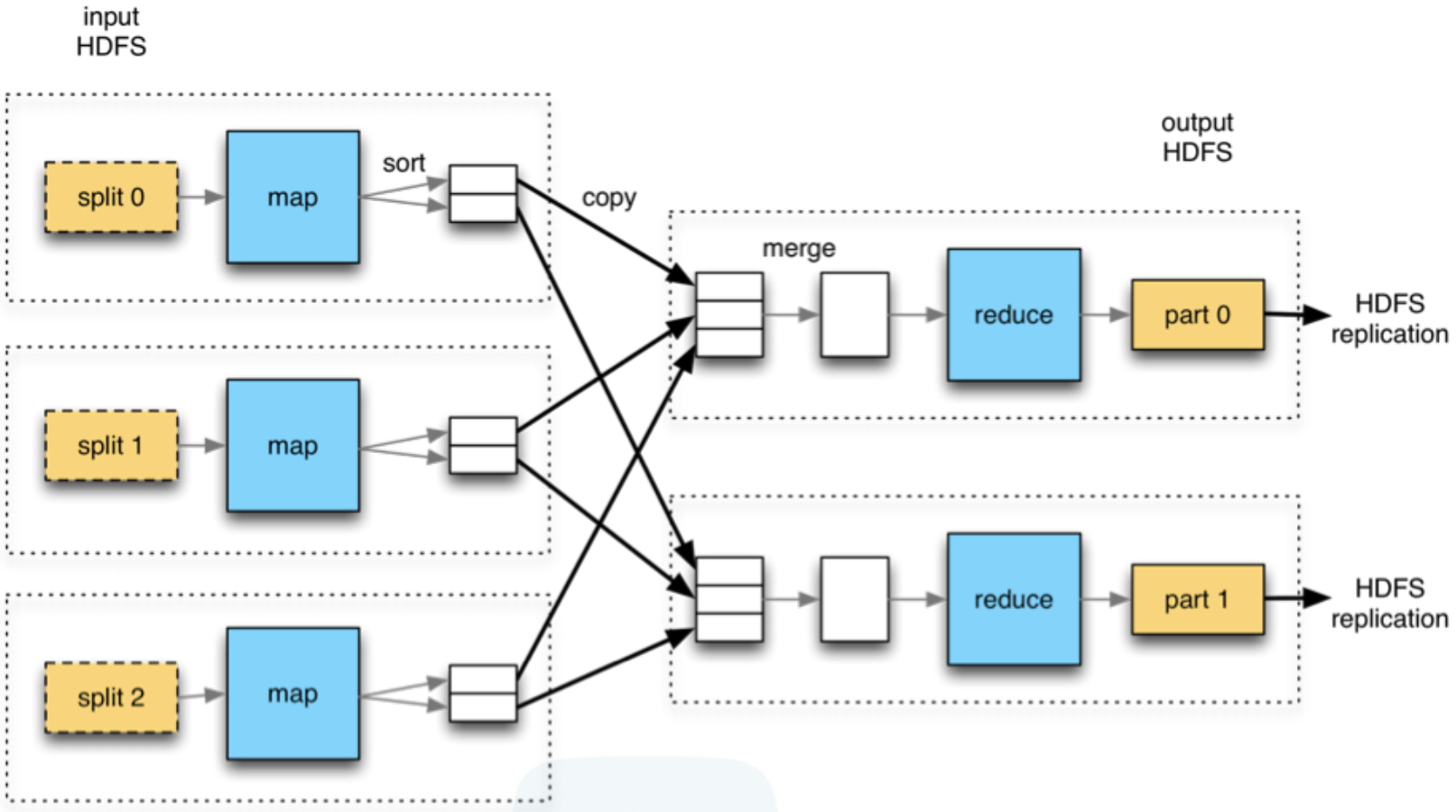
Hadoop MR简介

Google MapReduce

MapReduce

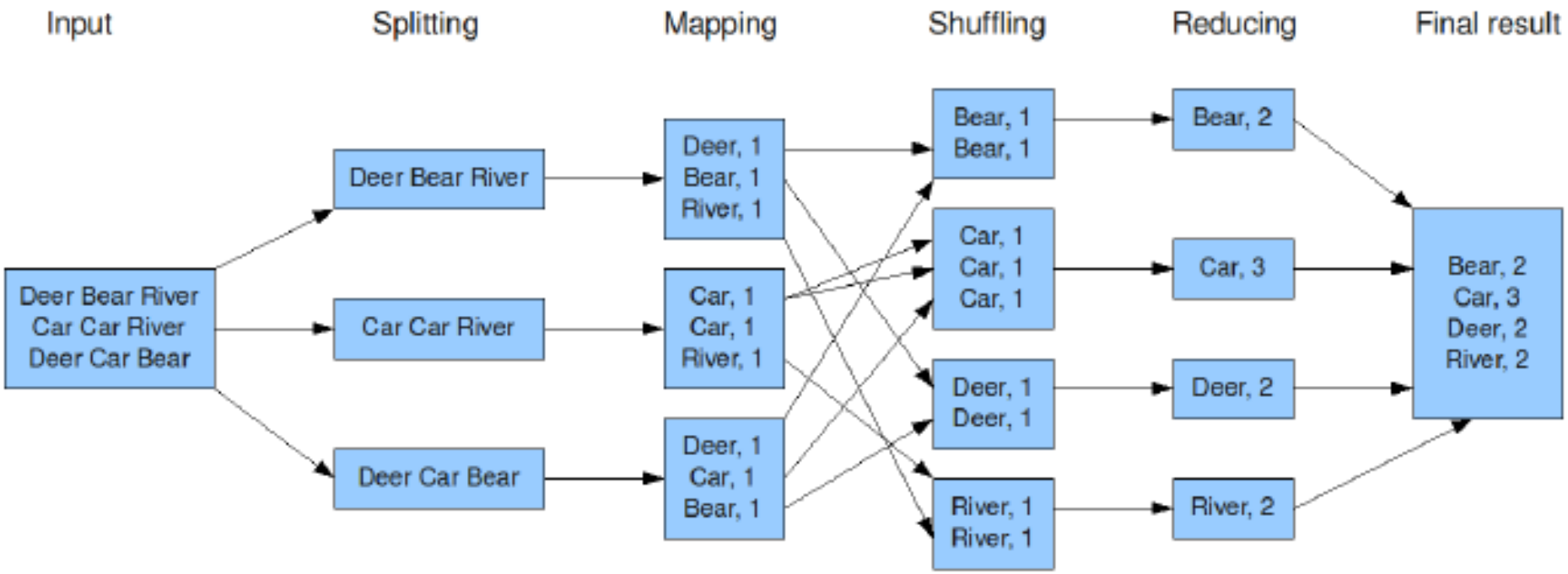
- 基于google MapReduce论文
- 只有map, reduce两个基本算子
- 只能按map -> reduce的逻辑单元执行
- 不同的MR任务之间，数据必须落地
- 不能在一个Job中进行较复杂的操作，例如：
  reduce -> map, reduce -> reduce,
  map-> map -> reduce

MapReduce简要流程



Hadoop The Definitive Guide

MR Word Count 示例



MR Word Count 示例

```
  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
  if (otherArgs.length < 2) {
    System.err.println("Usage: wordcount <in> [<in>...] <out>");
    System.exit(2);
  }
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  for (int i = 0; i < otherArgs.length - 1; ++i) {
    FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
  }
  FileOutputFormat.setOutputPath(job,
    new Path(otherArgs[otherArgs.length - 1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

02.

Hive

SQL & MR

Hive

- SQL -> MapReduce解释器
- 支持大部分常用查询语句、SQL2011标准
  - 聚合函数
  - Partition by
  - Window
  - Explode
- 支持很多内嵌函数
  - 字符串操作
  - 集合操作
  - 日期操作
  - JSON操作
  - UDF
- Cost-based decision
- Predicate pushdown

Hive vs 关系型数据库

	Hive	关系型数据库-mysql
范式	列类型支持Array, Map, Struct	默认基于第一范式
业务场景	离线数据统计	一般用于OLTP，也可用于OLAP
存储	元数据一般在关系型数据库中(MySQL)。自己没有数据存储功能，数据在HDFS中，依赖元数据库存储hdfs路径。支持的文件格式较多。	自己管理数据
常用操作	批量增，全量删，查	增删改查
事务	默认无事务	默认有事务

Hive vs MR

	Hive	MapReduce
工作效率	书写SQL	Coding + Unit Test
异常处理	SQL监测	靠个人对数据的理解程度
算子	支持大部分SQL查询，可以自定义UDF，UDAF，UDTF	主要支持Map和Reduce，其他算法都要先转化为MapReduce的逻辑去实现
运行效率	依赖HQL编写水平 自带执行计划、CBO、谓词下推等	依赖Coding水平，以及对业务逻辑的抽象能力。（可能会造成可读性下降）
计算复杂度	只要SQL可以实现即可，其他不需要关心	需要Coding处理各个MR之间的数据逻辑
数据管理	有元数据管理	手动管理数据逻辑

Hive Word Count 示例

```
SELECT word, count(1)
FROM
(SELECT
EXPLODE(SPLIT(text,' ')) AS word
FROM
document) AS t
GROUP BY word;
```

03.

Spark Core

比MR更高的抽象

Spark Core

基于RDD[T]这一抽象概念进行运算

- RDD的两个核心抽象属性
  - dependencies_（父依赖的RDD）
  - partitions_（数据的位置信息）

- RDD有多种算子
  - Transformation（几十种）
    - Map, filter, join, groupBy, mapPartitions
  - Action（至少十多种）
    - Collect, first, reduce, Count

Spark Core

- **RDD是不可变的**
  - Transformation -> 产生新的RDD实例
  - Action -> 得到基本类型等非RDD实例
    例如数组、每行数据实例（Int, String, Tuple, Class等）

- **RDD可以cache在缓存中**

- **Lazy Execution**
  - 把RDD组合成DAG（有向无环图）
  - 只有action会触发计算

Spark Core Transformation

	Arguments	Source	Return
map	f: T => U	RDD[T]	RDD[U]
filter	f: T => Boolean	RDD[T]	RDD[T]
flatMap	F: T => Seq[U]	RDD[T]	RDD[U]
groupByKey		RDD[(K,V)]	RDD[(K,Seq[V])]
reduceByKey	F: (V, V) => V	RDD[(K,V)]	RDD[(K,V)]
union		(RDD[T], RDD[T])	(RDD[T])
join		(RDD[K,V], RDD[K,W])	RDD[K,(V,W)]

Spark Core Action

	Arguments	Source	Return
count		RDD[T]	Long
collect		RDD[T]	Seq[T]
reduce	F: (T, T) => T	RDD[T]	T
saveAsTextFile	Path	RDD[T]	Unit
max		RDD[T]	T
first		RDD[T]	T

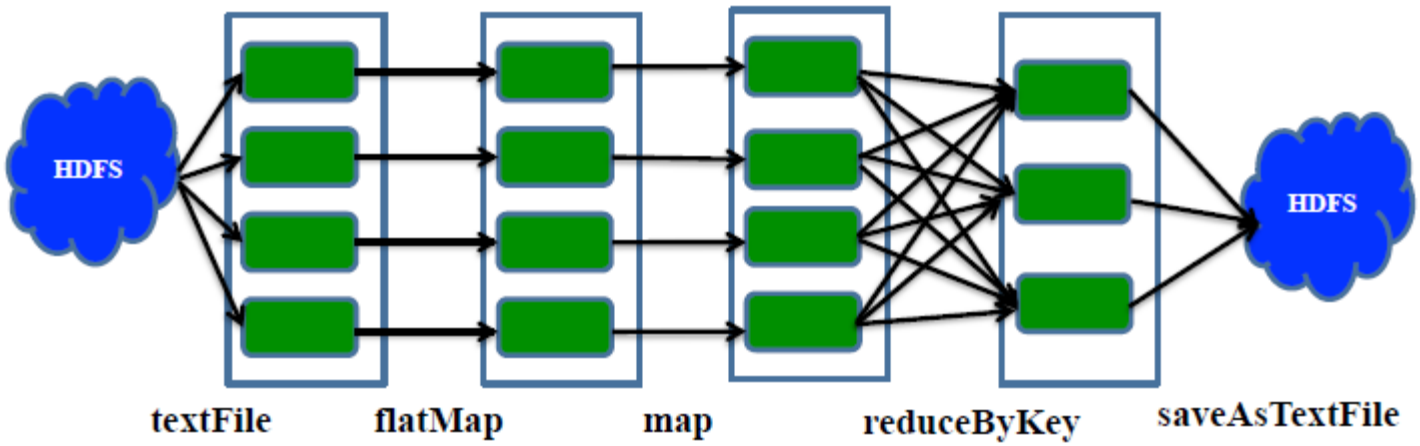
Spark Core vs MR

	Spark Core	MapReduce
代码书写	Scala(主要)、Java、Python	Java
算子	几十种不同算子	主要支持Map和Reduce，其他算法都要先转化为MapReduce的逻辑去实现
运行效率	Lazy 执行，一般比MR数据落地情况少，IO时间少	不同MR之间，必须数据落地
计算复杂度	只要可以关联成DAG图，就可以计算	需要Coding处理各个MR之间的逻辑
重复计算	如果有Cache，则可避免重复计算。	必须重新运行所需数据之前所有的MR

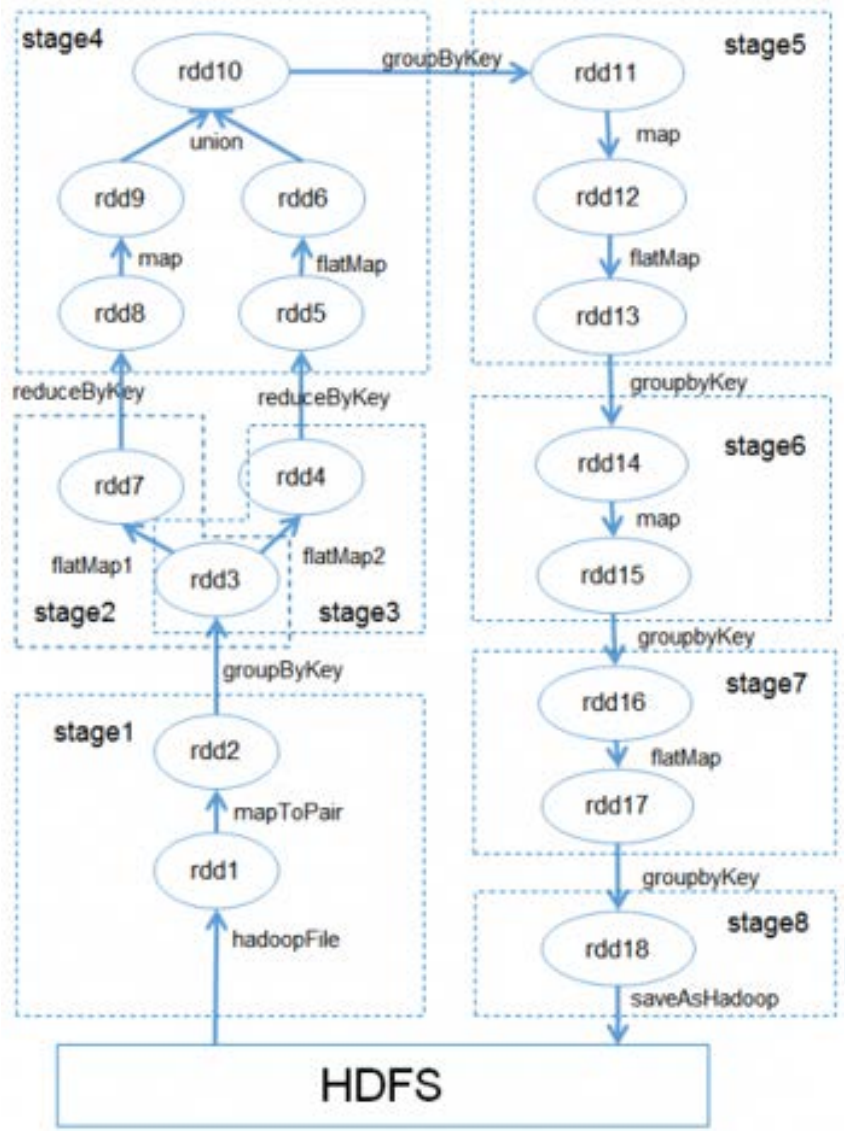
Spark Core Word Count 示例（1）

```scala
def main(args: Array[String]) {

  val spark = SparkSession
    .builder()
    .appName("Word Count")
    .config("spark.some.config.option", "some-value")
    .getOrCreate()

  val rowRdd = spark.sparkContext.textFile(args(1))

  val wcRdd = rowRdd.flatMap(line => line.split("\\s+")).
    map(word => (word,1)).reduceByKey((x,y) => x+y)
  /*
  val wc1Rdd = rowRdd.flatMap(_.split("\\s")).
    map((_, 1)).reduceByKey(_+_)
  */
  wcRdd.saveAsTextFile(args(2))
```

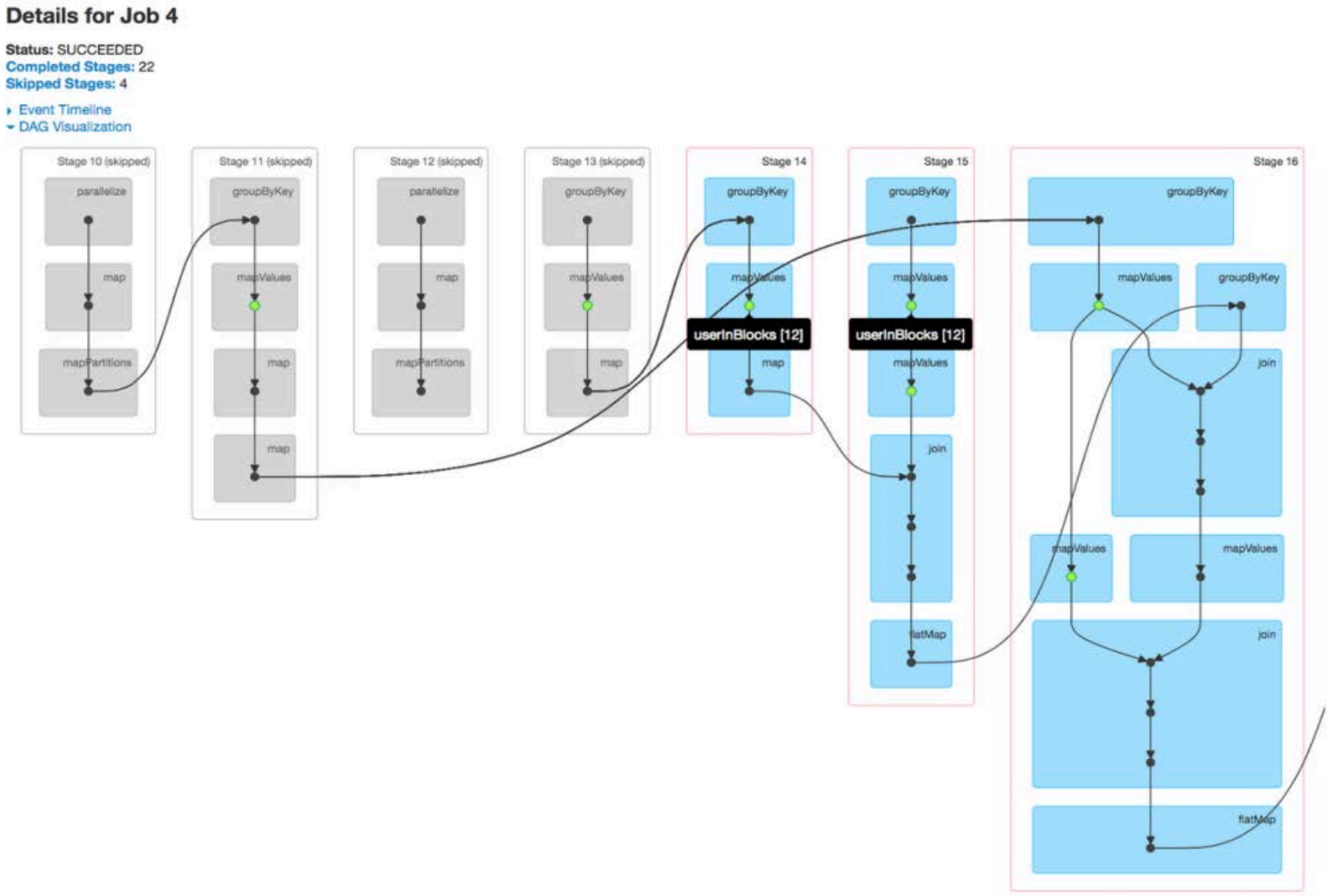
Spark Core Word Count 示例（2）



Spark DAG Shuffle



Spark DAG Shuffle



去哪儿?
Qunar.Com
聪 明 你 的 旅 行

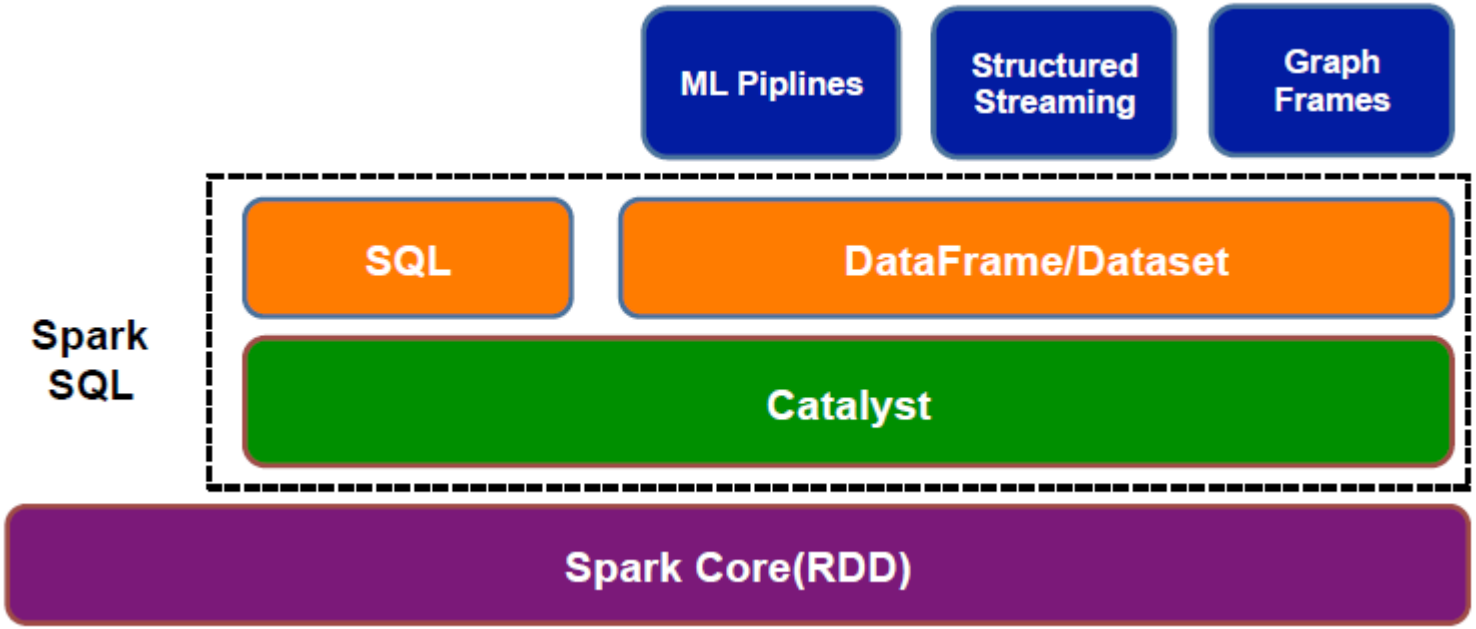
04.

Spark SQL

SQL & RDD

Spark SQL

Spark社区对待Spark SQL和Spark Core的态度



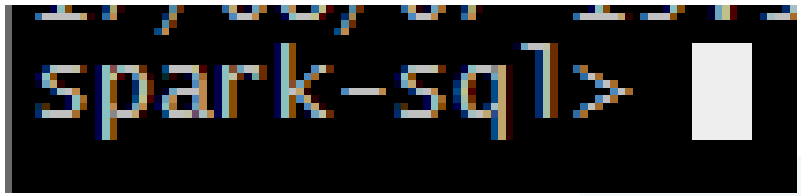
Spark SQL

基于Dataset、Column这两个抽象概念进行运算

Dataset = RDD[Row] + Schema

Schema = Array[] (name, dataType, nullable, metadata)

目前的Spark SQL是SQL和Coding的混合态



Spark SQL 数据源

1. 从hive中读取表
   val hiveDS = spark.table(“dbname.tablename”)
2. 从csv文件中读取
   val csvDS = spark.read.csv(“csv_file_path_1”,...)
3. 从json文件中读取
   val jsonDS = spark.read.json(“json_file_path_1”,...)
4. 通过jdbc读取
   val jdbcDS = spark.read.jdbc(url, tableName, properties)
   (参数可以只选取部分列，或者增加where条件。)
5. 从parquet文件、orc文件中读取
6. 从HBase、ES中读取
7. 从RDD转换(非结构化数据)

只要Hadoop支持读取的文件/数据，Spark都可以读取

Spark SQL 基本操作 Dataset(SQL)

- **Sql()**
  - **可以从表、临时表中通过sql语句，形成新的Dataset。**
- **createTempView()， createOrReplaceTempView ()**
  - **Dataset可以通过这两个方法，注册成临时表。然后通过sql()方法操作临时表，形成新的Dataset**
- **常用情景：**
  - **复用之前已有的SQL脚本，减少操作量。**

Spark SQL 基本操作 Dataset(Code)

- Spark SQL也可以“强行”分为Transformation(返回结果并不都是Dataset)和Action。
- 常用的Transformation就是各种Sql运算符，以及部分RDD的Transformation。

- join, sort, select, where(filter), groupBy, limit, union(unionAll), intersect, expect, sample, randomSplit, withColumn, dropDuplicates, drop, describe等

- Window.partitionBy().orderBy(),

- 常用的Action一般只使用show, count, first等

Spark SQL 基本操作 Column(Code)

- Column每一列的抽象 $"cName", 'cName, col("cName")
- 常用操作：avg(), count(), max(), cast(), as()等
- 比较操作：===, =!=, >, <, >=, <=等
- 条件判断：when().otherwise(), &&, ||, or, and等
- 空值判断：isnull()等
- 数字操作：+,plus(), -, minus(), *, /, %, rand(), sin()等
- 字符串操作，日期操作，数组操作，JSON操作等

去哪儿? Qunar.Com 聪明你的旅行

Spark SQL 基本操作演示

1. 通过SQL操作

```
spark.sql("SELECT a, b FROM TABLE_A")
  .cache().createOrReplaceTempView("tmp_1")

spark.sql(
  """SELECT b, c
    |FROM tmp_tbl_1 t1
    |LEFT JOIN TABLE_B t2
    |ON t1.a = t2.a
    |""".stripMargin)
  .cache().createOrReplaceTempView("tmp_2")

spark.sql(
  """
    |CREATE TABLE final_table AS
    |SELECT * FROM tmp_2
    |""".stripMargin).show()
```

Spark SQL 基本操作演示

2. 通过纯Coding操作

```
val tblADS = spark.table("TABLE_A").
  select("a","b").cache()
val tblBDS = spark.table("TABLE_B").cache()

val tmpDS = tblADS.as("t1").join(
  tblBDS.as("t2"),
  $"t1.a" === $"t2.a",
  "left").select($"t2.b".as("b"), $"t2.c".as("c"))

tmpDS.write.mode(SaveMode.Append).
  partitionBy("b").saveAsTable("final_table")
```

Spark SQL vs Hive执行性能

问题1:

现有A, B, C, D共4张表，需要分别得到A join B join C的结果，和A join B join D的结果。

问题2:

现有A(id, type), B(id, gender)共两张表，分别需要得到A JOIN B GROUP BY type的结果和A JOIN B GROUP BY gender的结果。

Spark SQL vs Hive书写效率

问题1：

表A有一百列，存在如”null”的脏数据（数据分析中，标准形式应该是null）。

Hive:

```
SELECT
CASE WHEN col1  = ‘null’ THEN null ELSE col1 END AS col1
…
FROM A;
```

Spark SQL vs Hive书写效率

```
def replaceEmptyAsNull(ds: Dataset[Row]) = {
  val replaceEmpty =
    for (StructField(name, dataType, _, _) <- ds.schema)
      yield {
    if(dataType.isInstanceOf[StringType]){
      (when(trim(col(name)) === "", null).
        otherwise(trim(col(name)))).as(name)
    }else{
      col(name)
    }
  }

  ds.select(replaceEmpty : _*)
}
```

Spark SQL vs Hive书写效率

问题2：

表A有一百列，找到所有null值率大于等于0.9的列名。

Hive:

每个列对is null进行聚合，除以count(1)，再挑出>=0.9的列。

Spark SQL vs Hive书写效率

```
def getNullCols(ds:Dataset[Row], lower:Double) = {
  val totalCnt = ds.cache().count()

  val aggCols =
    for (name <- ds.schema.fieldNames) yield {
    (sum(when(isnull(col(name)), 1).otherwise(0)) / totalCnt).
      as(name)
  }

  val agg = ds.agg(aggCols.head, aggCols.tail : _*)
  val aggVals = agg.first()

  for(col <- agg.schema.fieldNames
      if (aggVals.getAs[Double](col)) >= lower) yield col
}
```

Spark SQL vs Hive书写效率

问题3：

Json数据操作。

Hive:

Get_json_object()

Json_tuple()

Spark:

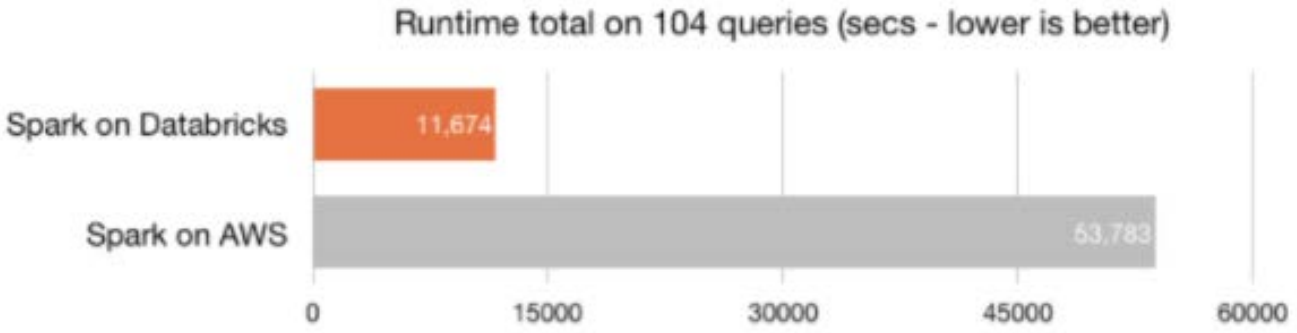
```
val jsonDS = spark.read.json(
spark.table("table_name").select("json_col").
map(_.getString(0)).rdd)
```

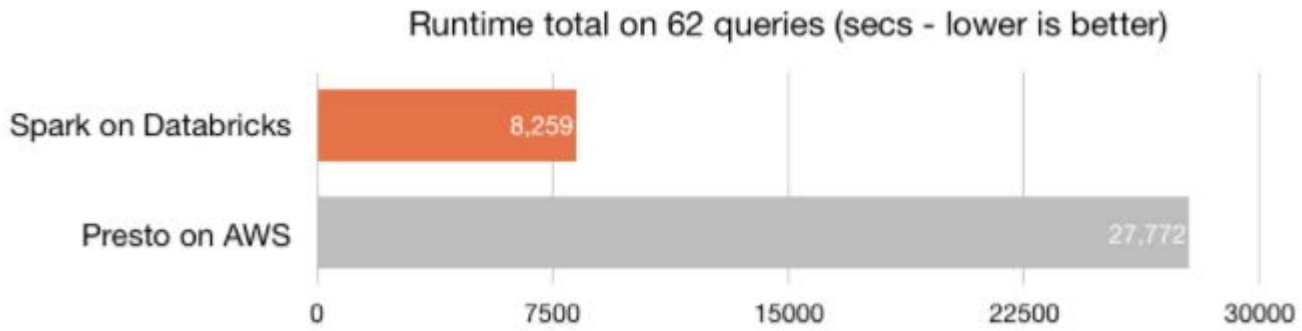
Hive vs Spark SQL

	Hive	Spark SQL
代码书写	写SQL 复制粘贴一大堆SQL	可以根据schema等信息进行coding，减少sql量。 因为是Lazy计算，所以很轻松可以把一个大SQL拆成很多小的Dataset。逻辑清晰。注释更方便（变量名是注释的一部分）。
算子	主要hive自带函数以及udf, udaf	Hive支持的基本都支持，还可以转换成rdd进行计算。
UDF,UDAF,UDTF	需要写jar包加载	可以直接在代码中注册。本身就是代码。
数据源	只能读取元数据中有的数据	多种多样，对部分数据源可以直接读取元数据，或自适应数据类型。 也可以从通过Spark Core的RDD把非结构化数据转化为Dataset。
计算效率	必须重新运行所需数据之前所有的SQL脚本。	同一个SparkSession中，如果有Cache，则可避免重复计算。 Spark Core比MR快。

Performance

TPC-DS v2.4 on Spark 3.0





Presto无法支持全部语句

FAQ

1. Spark SQL 存储数据到新表时，如何指定压缩格式与存储形式？

format(存储格式)
option(格式相关配置，比如压缩，比如csv的header，比如json的日期格式等)

举例：

```
ds.write.format("parquet").option("compression", "SNAPPY").saveAsTable("tbl")
```

去哪儿? Qunar.Com 聪明你的旅行

FAQ

Dataset使用parquet格式，snappy压缩

```
scala> val df1 = Seq((1, 2)).toDF("a", "b")
df1: org.apache.spark.sql.DataFrame = [a: int, b: int]

scala> df1.write.format("parquet").option("compression", "SNAPPY").saveAsTable("algorithm.tmptbl")

scala> spark.sql("desc formatted algorithm.tmptbl").show(100,false)
+----------------------------+----------------------------------------------------------------------------+-------+
|col_name                    |data_type                                                                   |comment|
+----------------------------+----------------------------------------------------------------------------+-------+
|a                           |int                                                                         |null   |
|b                           |int                                                                         |null   |
|                            |                                                                            |       |
|# Detailed Table Information|                                                                            |       |
|Database:                   |algorithm                                                                   |       |
|Owner:                      |                                                                            |       |
|Create Time:                |Tue Aug 22 11:03:50 CST 2017                                                |       |
|Last Access Time:           |Thu Jan 01 08:00:00 CST 1970                                                |       |
|Location:                   |hdfs://qunarcluster/user/[redacted]/hive/warehouse/algorithm.db/tmptbl      |       |
|Table Type:                 |MANAGED                                                                     |       |
|Table Parameters:           |                                                                            |       |
|  rawDataSize               |-1                                                                          |       |
|  numFiles                  |1                                                                           |       |
|  transient_lastDdlTime     |1503371030                                                                  |       |
|  totalSize                 |496                                                                         |       |
|  COLUMN_STATS_ACCURATE     |false                                                                       |       |
|  numRows                   |-1                                                                          |       |
|                            |                                                                            |       |
|# Storage Information       |                                                                            |       |
|SerDe Library:              |org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe                 |       |
|InputFormat:                |org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat               |       |
|OutputFormat:               |org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat              |       |
|Compressed:                 |No                                                                          |       |
|Storage Desc Parameters:    |                                                                            |       |
|  serialization.format      |1                                                                           |       |
|  compression               |SNAPPY                                                                      |       |
+----------------------------+----------------------------------------------------------------------------+-------+
```

FAQ

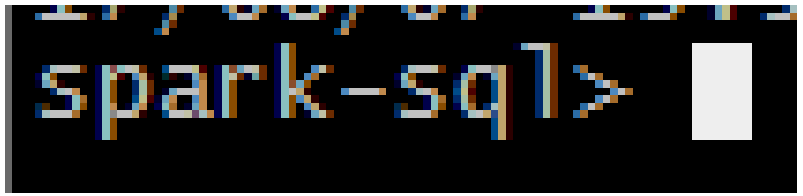
2. Spark SQL该怎么学？

想要系统、深入的学习的话，一般可以按照我PPT的顺序，前面的MR， hive打基础。然后基于Spark RDD之上的Spark SQL。

只是平时使用的话，推荐从spark.sql()这个方法开始用，因为这个方法可以适配基本上所有的hive sql语句（我还没见过不能用的hql逻辑）。

然后慢慢用一部分coding代替重复的sql逻辑。

如果不想coding，
只是单纯拿来代替hive就用
可以使用cache关键字。



当然，大家可以根据个人情况，混合上面三种方法一起用。

最后，spark官网是重点：Spark SQL
不会的可以qtalk找我沟通。

THANKS