DASCI

Instituto Andaluz Interuniversitario
en Ciencia de Datos e
Inteligencia Computacional

Ciencia de Datos a través del Big Data

Diego García (djgarcia@ugr.es)
Isaac Triguero (isaaktriguero@ugr.es)

Financiado por
la Unión Europea
NextGenerationEU

GOBIERNO
DE ESPAÑA

MINISTERIO
PARA LA TRANSFORMACIÓN DIGITAL
Y DE LA FUNCIÓN PÚBLICA

SECRETARÍA DE ESTADO
DE DIGITALIZACIÓN
E INTELIGENCIA ARTIFICIAL

Plan de
Recuperación,
Transformación
y Resiliencia

UNIVERSIDAD
DE GRANADA

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

REPSOL

CAMBRIDGE UNIVERSITY PRESS

Chapter 5

Spark SQL II

© Isaac Triguero and Mikel Galar

Learning Outcomes

- The importance of structured data to optimize Big Data processing [KU]
- The concept of DataFrames to work with structured data in Spark [KU]
- How to operate with DataFrames: understanding the differences with RDDs [KU, IS, PPS]
- How to use DataFrames in practice for problem solving [IS, PPS]

Recap/Summary

- RDDs are very good but they are so flexible that Spark can't apply further optimizations
- From unstructured to structured data with two projects: Tungsten (memory) and Catalyst (execution).
- **Key idea**: Imposing structure on your data allows Spark to avoid Java serialization and object overhead
- We will focus on the DataFrame API for Python
  - Same performance in all languages! As 'Catalyst' translates everything to the same code in Java!

Outline

- Structured data with Apache Spark
  - Basic concepts
  - DataFrames
  - Basic operations
  - Advanced concepts

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- A **new entry point** to Spark: `SparkSession`
  - Adds new features (Hive and SparkSQL)
  - Unifies all the different contexts in Spark (i.e., `sqlContext`, `SparkContext`, `StreamingContext` and `HiveContext`).
  - We can still access the `SparkContext`

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .master("local[*]") \
    .appName("Chapter 5: pyspark SQL") \
    .getOrCreate()

sc = spark.sparkContext
```

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- What is a **DataFrame**?
  - A **2-D data structure** (a table), **organized by columns** defined in a schema
    - All elements of a column belong to the same **data type**
    - **Column name** is used to refer to the column
  - In Spark, a DataFrame is a *distributed collection of elements of type* `Row`
  - A `Row` is an array that can be indexed by name (like a dictionary)
  - Like RDDs, they are **immutable**!

```
# +----+-------+
# | age|   name|
# +----+-------+
# |null|Michael|
# |  30|   Andy|
# |  19| Justin|
# +----+-------+
```

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- Creating a DataFrame
  - From an existing RDD, Hive Table or any other Spark data source
  - Requires a **schema** defining the column names and their types
    - Also, whether they can take NULL values or not
      - Similar to a `CREATE TABLE` in SQL
- Three ways to **define the schema**
  - Infer it automatically **from the data** (e.g., JSON files, RDDs, etc)
  - Infer it automatically **from metadata** (e.g., JDBC, JavaBeans)
  - **Explicit definition**

SparkSQL: Basic Concepts

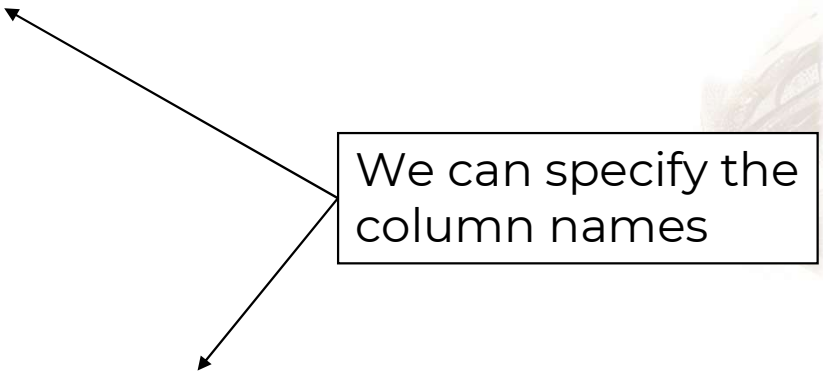
CAMBRIDGE UNIVERSITY PRESS

- **Inferring the schema from the data**
  - **From tuples**

```
>>> l = [('Alice', 1)]
>>> spark.createDataFrame(l).collect()
[Row(_1=u'Alice', _2=1)]
>>> spark.createDataFrame(l, ['name', 'age']).collect()
[Row(name=u'Alice', age=1)]
```

  - **From an RDD of tuples**

```
>>> rdd = sc.parallelize(l)
>>> spark.createDataFrame(rdd).collect()
[Row(_1=u'Alice', _2=1)]
>>> df = spark.createDataFrame(rdd, ['name', 'age'])
>>> df.collect()
[Row(name=u'Alice', age=1)]
```



SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- **Inferring the schema from the data**
  - **From RDDs of Rows**

```
>>> from pyspark.sql import Row
>>> rdd = sc.parallelize([('Alice', 1)])
>>> Person = Row('name', 'age')
>>> person = rdd.map(lambda r: Person(*r))
>>> df2 = spark.createDataFrame(person)
>>> df2.collect()
[Row(name=u'Alice', age=1)]
```

  - **From Pandas and to Pandas**

```
>>> spark.createDataFrame(df.toPandas()).collect()
[Row(name=u'Alice', age=1)]
>>> spark.createDataFrame(pandas.DataFrame([[1, 2]])).collect()
[Row(0=1, 1=2)]
```

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- **Specifying the schema**
  - You will normally try to do this automatically, but if you need to specify it manually, you can use `StructType`.
  - You need to access the different SQL types of Spark.

```
>>> from pyspark.sql.types import *
>>> schema = StructType([
...     StructField("name", StringType(), True),
...     StructField("age", IntegerType(), True)])
>>> df3 = spark.createDataFrame(rdd, schema)
>>> df3.collect()
[Row(name=u'Alice', age=1)]
```

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- **Schema When Reading/Writing from/to Files**
  - You can **read from different data types** (JSON, CSV, text) and infer their schema, e.g.,

```
spark.read.json("data/data.json")
```

  - You can also **write DataFrames directly to JSON, CSV or text**, e.g.,

```
json_df.write.json("data/json_file.json")
```

    - As with RDDs, the output Will be a directory

SparkSQL: Basic Concepts

- **Some remarks**
  - Use `show()` instead of `collect()` for better looking output if you just want to see the content of the DataFrame
  - Inferring a schema for a large dataset might be too slow. **Use a random subset of the data to do this**

```
spark.read.option("samplingRatio", 0.2).json("data/data.json")
```

  - **DataFrame to RDD to DataFrame**. Both data structure are **interoperable**:

```
>>> df.rdd
>>> spark.createDataFrame(rdd)
```

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- The column names are like indexes but doing something like `df["colName"]` will not provide the actual content, but just the column name. **You need to perform an operation to select the column**
- Again, we have two types of operations:
  - **Transformations**: Create a new Data Frame (lazily)
  - **Actions**: Return the result to the driver
- All of them use Catalyst to optimize computations!

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- **Transformations**
  - Similar to SQL type of operations

Transformation	Meaning
`select(*cols)`	Returns a new DataFrame using a series of expressions that could be column names or `Column` objects. If `*` is used, all the columns of the DataFrame are returned.
`selectExpr(*expr)`	Variation of `select` to allow for SQL expressions in String format.
`filter(condition) / where(condition)`	Filter the rows according to a condition.
`orderBy(*cols,**kwargs) / sort(*cols, **kwargs)`	Returns a new DataFrame sorted by the specified columns (in ascending order by default).

Let's try them hands-on!

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- In SparkSQL there are **many equivalent ways** to program operations

```
>>> df.sort(df.age.desc()).collect()
[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]

>>> df.sort("age", ascending=False).collect()
[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]

>>> df.orderBy(df.age.desc()).collect()
[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]
```

Equivalent

```
>>> from pyspark.sql.functions import *
>>> df.sort(asc("age")).collect()
[Row(age=2, name=u'Alice'), Row(age=5, name=u'Bob')]

>>> df.orderBy(desc("age"), "name").collect()
[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]

>>> df.orderBy(["age", "name"], ascending=[0, 1]).collect()
[Row(age=5, name=u'Bob'), Row(age=2, name=u'Alice')]
```

Equivalent

- **Additional Transformations**

Transformation	Meaning
`distinct()`	Returns a new DataFrame with the unique rows of the original one.
`dropDuplicates(*cols)`	Returns a new DataFrame without duplicates only considering the columns specified.
`withColumn(colName, col)`	Returns a new DataFrame adding a new column or replacing an existing column with the given name.
`withColumnRenamed(existing, new)`	Returns a new DataFrame renaming an existing column.
`drop(col)`	Returns a new DataFrame without a specified column.
`limit(num)`	Limits the number of rows obtained as a result.
`cache()`	Keeps the DataFrame cached for future re-use.

Let's try them hands-on!

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- **Transformations: Column operations**
  - Very useful to operate at the column level (not at the DataFrame level!)
  - These are functions we can use **INSIDE** of transformations
  - These functions return **column expressions**
    - Same problem as we showed before when doing: `df['colName']`
  - They **don't do anything directly**, you need to put them in a transformation (e.g., a `select`)
  - Example of these: `alias()`, `between()`, `when()`, `explode()`…

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- **Transformations: Column operations**

Operation	Description
`alias(*alias)`	Returns a column with a new name.
`between(lowerBound, upperBound)`	True if the value is in the range specified.
`isNull() / isNotNull()`	True if the value is NULL or not.
`when(condition, value) / otherwise(value)`	Evaluates a list of conditions and performs a transformation based on those.
`length(col)`	Returns the length of the column.
`substring(startPos, len), like(other), startswith(other), endswith(other)`	Functions to operate with strings.
`expr(str)`	Parses the input string into the column that it represents.
`isin(*cols)`	True if the value is in the list of arguments.
`lit(value)`	Creates a column with value provided.
`explode(col)`	Returns a new row for each element of the array.

Let's try them hands-on!

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- **Transformations with Pseudo-Sets**
  - Similar to RDDs, we can also do set-like operations

Transformation	Description
`distinct()`	Returns a new DataFrame with the unique rows of the original one.
`union(df)`	Returns the union of elements of two DataFrames (keep duplicates).
`intersect(df)`	Returns the union of elements of two DataFrames (removes duplicates).**Warning: This requires a Shuffle!**
`subtract(df)`	Returns a DataFrame with the elements present in the first DataFrame, but not in the second one. **Warning: This requires a Shuffle!**
`crossJoin(df)`	Returns a DataFrame with the cartesian product of both DataFrames (all possible pairs of rows of both).

Let's try them hands-on!

SparkSQL: Basic Concepts – Aggregations

CAMBRIDGE UNIVERSITY PRESS

- **Aggregations and operations on grouped DataFrames**
  - Similar to the descriptive statistics methods for numeric RDDs, there are specific operations that allow to **aggregate numeric columns** of a DataFrame
    - E.g., to compute `avg`, `max`, `min`, `sum` or `count`
  - You can first group your data in a certain way prior to perform the aggregation (using `groupBy`)
    - E.g., you want to count the number of people with the same name
    - `groupBy` provides a different `DataType` (**`GroupedData`**) for which we can apply different transformations (e.g., `count`, `max`, `aggregations`, etc.)

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- **Transformations to Perform Aggregations**
  - Aggregations for numerical data and grouped data

Transformation	Description
agg(*exprs)	Performs aggregations over the entire DataFrame – exprs can be a dictionary or list of expression.
groupBy(*cols)	Groups a DataFrame using the specified columns (GroupedData) to perform aggregations over those groups.

Let's try them hands-on!

- **Transformations to Perform Aggregations**
  - Aggregations for `GroupedData`

Transformation	Description
`avg(*cols)` / `mean(*cols)`	Calculates the average value for each group in each numeric column.
`count()`	Counts the number of rows for each group.
`max(*cols)`	Calculates the max for each group in each numeric column specified.
`min(*cols)`	Calculates the min for each group in each numeric column specified.
`sum(*cols)`	Calculates the sum for each group in each numeric column specified.
`pivot(pivot_col, values)`	Pivots over a column of the DataFrame to perform a specific aggregation.
`agg(*exprs)`	Same as before, but to each group of the DataFrame.

Let's try them hands-on!

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- **Join-like SQL Transformations**
  - Obviously, we can perform join-like transformations

Transformation	Description
`join(other, on, how)`	Joins two DataFrames.

`other`: The other DataFrame for the join.
`on`: name of the column on which to perform the join. It can also be a list of columns or a Column expression.
`how`: type of join. The most common are `'inner'` (default), `'outer'`, `'left_outer'` and `'right_outer'`.

Let's try it hands-on!

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- **Actions**: return a result to the driver

Action	Description
`show(n=20, truncate=True)`	Prints n rows of the DataFrame. Truncate indicates whether strings should be cut if too long.
`count()`	Returns the number of rows in the DataFrame.
`collect()`	Returns all the rows of the DataFrame as a list of `Rows`. **Warning: Should fit in the driver's main memory**
`first()`	Returns the first `Row` of the DataFrame.
`take(n)`	Returns the first n rows of the DataFrame as a list of `Row` elements.
`toPandas()`	Returns the content of the DataFrame as a pandas.DataFrame. **Warning: Should fit in the driver's main memory**
`columns`	Returns the column names of the DataFrame as a list.
`describe(*cols)`	Provides some statistics for numeric columns (count, mean, stddev, min, and max).
`explain(extended=False)`	Prints out the physical and logical plans for debugging.

Let's try them hands-on!

SparkSQL: Basic Concepts

CAMBRIDGE UNIVERSITY PRESS

- **Caching DataFrames**: it doesn't happen by default!!

```
Lines_df = spark.read.text("...")
words_df = lines_df.select(
                explode(split('value', ' ')).alias('word')
           ).cache()
words_df.where(words_df.word.like("%block-head%")).count()
words_df.where(words_df.word.like("%spear%")).count()
```

SparkSQL– Advanced Concepts – User Defined Functions

CAMBRIDGE UNIVERSITY PRESS

- SparkSQL allows for User Defined Functions (UDFs) to transform a DataFrame.

```
>>> from pyspark.sql.functions import udf
>>> from pyspark.sql.types import IntegerType
>>> slen = udf(lambda s: len(s), IntegerType())

>>> df.select(slen(df.name).alias('slen')).collect()
[Row(slen=5), Row(slen=3)]
```

- Note that **these functions won't be as efficient** as the native functions provided by Apache Spark
  - Although there are being improvements to apply vectorized operations since Spark 2.3 and 3.0

SparkSQL– Advanced Concepts – Using SQL

- We can apply **SQL queries** on a DataFrame
  - They follow the same optimization (Catalyst, Tungsten)
  - Return a DataFrame
  - The only thing you need to d o is to ‘register’ a Data Frame as a SQL temporary view

```
# Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")

sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
```

SparkSQL– Advanced Concepts – Pandas on Spark

CAMBRIDGE UNIVERSITY PRESS

- **Pandas DataFrame API on top of Spark** (initially 'koalas')
  - **Fully integrated** with Spark since 3.2
    - Can improve pandas even on a single machine (Catalyst + multithread optimizations)
  - **Simple** to use

```
import pyspark.pandas as ps
```

  - From Spark DataFrames to pandas-on-Spark DataFrames

```
people_ps = people_df.pandas_api()
```

  - Operating with pandas-on-Spark is **almost the same as doing it with pandas**

SparkSQL– Quick test

CAMBRIDGE UNIVERSITY PRESS

- You need to make sure that you know where operations are taking place.
- To concatenate two DataFrames, you could do this:

```
>>> a = aDF.collect()
>>> b = bDF.collect()
>>> cDF = sqlContext.createDataFrame(a + b)
```

- What happens if `aDF` is too big?
  - The driver may not have enough memory
  - It will take a long time to bring everything to the driver
- Where is `a + b` happening? Locally in the driver? Or Distributed?

SparkSQL– A few caveats

- A few things to always remember:
  - Always use the API from SparkSQL to perform operations
  - Do not use `collect()` when handling big datasets!
    - Use `take(n)`
  - Cache your DataFrames!

Take-home message

- Hands—on session on Spark SQL: Creating DataFrames, Transformations and Actions, and Advanced concepts
- DataFrames have a schema: inferred, from metadata or explicitly defined
- Operating with DataFrames is similar to working with SQL
- DataFrames are interoperable with RDDs and share many characteristics (Transformations and Actions)
- Spark SQL provides highly optimized functions. Try to stick to them

Next lecture:

- Machine Learning with Big Data

To Learn More

CAMBRIDGE UNIVERSITY PRESS

- **Spark SQL**
  - A tale of three Spark APIs
  - Spark SQL paper
  - The Dataset API
  - More about UDFs
- **Python**
  - The Pandas tutorial

CAMBRIDGE UNIVERSITY PRESS

Chapter 5

Spark SQL II

© Isaac Triguero and Mikel Galar