Learning Spark

Spark’s Structures APIs

Master’s degree, Universitat de Lleida

/bluetab
an IBM Company

REQUIREMENTS

- Download this file
- Open a google collaboratory
- Upload the file to your Google Drive

```
[1] from google.colab import drive
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tru
```

```
[2] !python /content/drive/MyDrive/UDL/install_pyspark.py

    Install JAVA 8
    Collecting wget
      Downloading wget-3.2.zip (10 kB)
    Building wheels for collected packages: wget
      Building wheel for wget (setup.py) ... done
      Created wheel for wget: filename=wget-3.2-py3-none-any.whl size=9675 sha256=e479514f9a0f7182d801131848b87a5359603f5ae510cb6
      Stored in directory: /root/.cache/pip/wheels/a1/b6/7c/0e63e34eb06634181c63adacca38b79ff8f35c37e3c13e3c02
    Successfully built wget
```

/content/drive/MyDrive/<path_to_file.py>

This, will prepare the environment in google drive to work with PySpark on it.

ÍNDICE

What's under an RDD?

- RDD is the most basic abstraction in Spark.
- Vital characteristics:
  - Dependencies
    - instruct Spark HOW an RDD is build, with its inputs as required. This gives RDDs **resiliency**.
  - Partitions (with some locality information)
    - It provides Spark the ability to split the work to parallelize computation on partitions across executors.
  - Compute function : Partition => Iterator[T]
    - It produces an Iterator[T] for the data that will be stored in the RDD.

PROBLEMS:

- The computation is opaque to Spark
- The Iterator[T] data type is also opaque for Python RDDs
- Spark has no way to optimize the expression
- Spark has no knowledge of the specific data type in T.

Structuring Spark

Spark 2.X = 🧡

- Key schemes:
  - To express computations by using common patterns found in data analysis (high-level operations, as filtering, selecting, counting, aggregations, averaging, and grouping) by using a DSL.
  - To allow programmers to organize our data in a tabular format, with supported data types
- Benefits:
  - Expressivity
  - Simplicity
  - Uniformity

Structuring Spark

Spark 2.X = 🧡

RDD EXAMPLE:

```
# Create an RDD of tuples (name, age)
dataRDD = sc.parallelize([("Brooke", 20), ("Denny", 31), ("Jules", 30), ("TD", 35),
("Brooke", 25)])
# Use map and reduceByKey transformations with their lambda
# expressions to aggregate and then compute average
agesRDD = (dataRDD
        .map(lambda x: (x[0], (x[1], 1)))
        .reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
        .map(lambda x: (x[0], x[1][0]/x[1][1])))
```

DF API EXAMPLE (PYTHON):

```
# Create a DataFrame
data_df = spark.createDataFrame([("Brooke", 20), ("Denny", 31), ("Jules", 30), ("TD", 35),
("Brooke", 25)], ["name", "age"])
# Group the same names together, aggregate their ages, and compute an average
avg_df = data_df.groupBy("name").agg(avg("age"))
```

EXAMPLE IN COLLAB

DF API EXAMPLE (SCALA):

```
// Create a DataFrame of names and ages
val dataDF = spark.createDataFrame(Seq(("Brooke", 20), ("Brooke", 25),
 ("Denny", 31), ("Jules", 30), ("TD", 35))).toDF("name", "age")
// Group the same names together, aggregate their ages, and compute an average
val avgDF = dataDF.groupBy("name").agg(avg("age"))
```

Structuring Spark

The DataFame API

- Inspired by pandas DataFrames in structure, format and a few specific operations
- Are like distributed in-memory tables with named columns and schemas, where each column has a specific data type: integer, string, array, map, real, date, timestamp, etc.
- Like tables, for human eye.
- Are inmutable.

Structuring Spark

Spark's Basic Data Types

DataType	Value assigned in Scala	Value assigned in Python	API to instantiate
Byte type	Byte	int	DataTypes.ByteType
ShortType	Short	int	DataTypes.ShortType
IntegerType	Int	int	DataTypes.IntegerType
LongType	Long	int	DataTypes.LongType
FloatType	Float	float	DataTypes.FloatType
Double	Double	float	DataTypes.Double
BooleanType	Boolean	bool	DataTypes.BooleanType
StringType	String	str	DataTypes.StringType
DecimalType	java.math.BigDecimal	decimal.Decimal	DataTypes.DecimalType

Structuring Spark

Spark’s Structured and Complex Data Types

DataType	Value assigned in Scala	Value assigned in Python	API to instantiate
BinaryType	Array[Byte]	bytearray	DataTypes.BinaryType
TimestampType	java.sql.Timestamp	datetime.datetime	DataTypes.TimestampType
DateType	java.sql.Date	datetime.date	DataTypes.DateTime
ArrayType	scala.collection.Seq	list, tuple or array	DataTypes.createArrayType(ElementType)
MapType	scala.collection.Map	dict	DateTypes.createMapType(KeyType, ValueType)
StructType	scala.collection.spark.sql.Row	list or tuple	StructType(ArrayType[FiledTypes])
StructField	A value corresponding to the type of this field	A value corresponding to the type of this field	StructField(name, dataType, [nullable])

Structuring Spark

Schemas and Creating DataFrames

- An schema defines the column names and associated data types for a DataFrame.
- Benefits of schema-on-read approach:
  - It frees Spark from the responsibility of inferring data types.
  - Prevents Spark from creating a separate job just to read a large part of your file to determine the schema
  - Programmers can catch errors early if the data does not match the schema.
- RECOMMENDATION: define the schema if we are reading large files from a data source.
- LET'S CODE!

Structuring Spark

Columns and expressions

- Named columns in DataFrames are conceptually similar to named columns in pandas or R DataFrames, they describe a type of field.
- We can list all the columns by their names, perform operations on their values using relational or computational expressions, also mathematical expressions...
- In Spark’s supported languages, columns are objects with public methods (represented by the Column type).
- [Scala/Java] We can also use the col() function, which returns a Column object
- [Python] We can also use df[‘<column_name>’] , which returns a Column object
- LET’S CODE!

Structuring Spark

Rows

- A row in Spark is a generic Row object. containing one or more columns.
- Each column may be of the same data type or different.
- As a Row is an object of Spark, we can instantiate a Row in each of Spark's supported languages and access its fields by an index, starting at 0.
- LET'S CODE!

Structuring Spark

Transformations and actions (page 28, chapter 2)

Spark operations can be classified in two types: transformations and actions

- Transformations: transform a DF into a new DF (select, filter. orderBy, groupBy, join)
  - All transformations are evaluated lazily
- An action (show, take, count, collect, save) triggers the lazy evaluation of all the recorded transformations
- LET'S CODE!

Structuring Spark

Projections and Filters

- A projection in relations parlance is a way to return only the rows matching a certain relations condition, by using filters.
- In spark, projections are done by select() and filters by filter() or where() methods.
- LET'S CODE!

RECOMMENDATION

- Read the section related with the Catalyst (in chapter 6 we are going to talk more about de Spark core optimizers)

¡Gracias!

alba.lamas@bluetab.net

¡Síguenos!

https://bluetab.net/

https://www.linkedin.com/company/bluetab/