Higher level data processing in Apache Spark

Pelle Jakovits

12 October, 2016, Tartu
Outline

• Recall Apache Spark
• Spark DataFrames
  – Introduction
  – Creating and storing DataFrames
  – DataFrame API functions
• SQL in Spark
• Spark MLlib
Recalling Apache Spark

• Supports in-memory computing
• Spark works with Scala, Java, Python and R
• Integrated with Hadoop Yarn and HDFS
• Uses Resilient Distributed Datasets to abstract data that is to be processed
• Massively scalable distributed data processing framework
  – All Spark code is automatically parallelized
• Fault tolerant
Frameworks powered by Spark

Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)

Apache Spark
Spark DataFrame

• A DataFrame is a Spark Dataset organized into named columns.
• Equivalent to a table in a relational database or a data frame in R or Python.
  – Optimized under the hood
  – Operations applied to Spark DataFrames are inherently parallel
• DataFrames can be constructed from a wide array of sources
  – Structured data files (json, csv, …)
  – Tables in Hive, external databases
  – Existing Spark RDDs.
DataFrame Example

textFile = sc.textFile("hdfs://...")

# Creates a DataFrame having a single column named "line"
df = textFile.map(lambda r: Row(r)).toDF(["line"])
errors = df.filter(col("line").like("%ERROR%"))

# Counts all the errors
errors.count()

# Counts errors mentioning MySQL
errors.filter(col("line").like("%MySQL%")).count()

# Fetches the MySQL errors as an array of strings
errors.filter(col("line").like("%MySQL%")).collect()
Creating DataFrames

- Reading data From JSON file:
  \[
  \text{val df} = \text{spark.read.json}(/mnt/data/people.json)
  \]

- Reading data From CSV file:
  \[
  \text{val songs} = \text{spark.read.format("csv").option("header", "true").load(/mnt/data/Top_1_000_Songs_To_Hear_Before_You_Die.csv)}
  \]

- Reading data From parquet file inside SQL statement:
  \[
  \text{val sqlDF} = \text{spark.sql("SELECT * FROM parquet./mnt/data/users.parquet")}
  \]

- Creating from an existing RDD object:
  \[
  \text{val df} = \text{linesRDD.toDF()}
  \]
DataFrame content

scala> spark.read.format("csv").option("header", "true").load("/mnt/data/Top_1_000_Songs_To_Hear_Before_You_Die.csv")

scala> songs.show(5)

+------------------------------------------+-------------------+-----------------+---+----------------------------------------------------+
| THEME | TITLE           | ARTIST          | YEAR | SPOTIFY_URL                                        |
+------------------------------------------+-------------------+-----------------+---+----------------------------------------------------+
| Love | The Look of Love | ABC             | 1982 | http://open.spotify.com/playlist/kETRJXxthWteOApQW3S3F2 |
| Love | God Only Knows   | The Beach Boys  | 1966 | http://open.spotify.com/playlist/kETRJXxthWteOApQW3S3F2 |
| Love | Good Vibrations  | The Beach Boys  | 1966 | http://open.spotify.com/playlist/kETRJXxthWteOApQW3S3F2 |
| Love | Wouldn't It Be Nice | The Beach Boys | 1966 | http://open.spotify.com/playlist/kETRJXxthWteOApQW3S3F2 |
+------------------------------------------+-------------------+-----------------+---+----------------------------------------------------+

only showing top 5 rows
Structure of the DataFrame

```
scala> songs.printSchema()

root
 |-- THEME: string (nullable = true)
 |-- TITLE: string (nullable = true)
 |-- ARTIST: string (nullable = true)
 |-- YEAR: string (nullable = true)
 |-- SPOTIFY_URL: string (nullable = true)
```
Select

scala> songs.select("THEME", "ARTIST").show(5)

+---------------+----------------------+
| THEME | ARTIST |
+---------------+----------------------+
| Love | ABC |
| Love | Badly Drawn Boy|
| Love | The Beach Boys|
| Love | The Beach Boys|
| Love | The Beach Boys|
+---------------+----------------------+
only showing top 5 rows
### Filter

```scala
scala> songs.filter("year < 1930").show(5)
```

<table>
<thead>
<tr>
<th>THEME</th>
<th>TITLE</th>
<th>ARTIST</th>
<th>YEAR</th>
<th>SPOTIFY_URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Love</td>
<td>Let's Do It, Let?...</td>
<td>Cole Porter</td>
<td>1928</td>
<td>null</td>
</tr>
<tr>
<td>People and places</td>
<td>Jerusalem</td>
<td>William Blake, Ch...</td>
<td>1916</td>
<td>null</td>
</tr>
<tr>
<td>People and places</td>
<td>Things Are Worse ...</td>
<td>Sam Mayo</td>
<td>1922</td>
<td>null</td>
</tr>
<tr>
<td>People and places</td>
<td>Stack O? Lee</td>
<td>Mississippi John</td>
<td>1928</td>
<td><a href="http://open.spotify.com/track/6Yc4e4IqR6y72KcM9E7Z2K">http://open.spotify.com/track/6Yc4e4IqR6y72KcM9E7Z2K</a></td>
</tr>
<tr>
<td>People and places</td>
<td>T for Texas</td>
<td>Jimmie Rodgers</td>
<td>1928</td>
<td><a href="http://open.spotify.com/track/6Yc4e4IqR6y72KcM9E7Z2K">http://open.spotify.com/track/6Yc4e4IqR6y72KcM9E7Z2K</a></td>
</tr>
</tbody>
</table>

only showing top 5 rows
Grouping and Aggregating

```scala
scala> songs.groupBy("YEAR", "THEME").count().orderBy(desc("count")).show(5)
```

---
<table>
<thead>
<tr>
<th>YEAR</th>
<th>THEME</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1979</td>
<td>Party songs</td>
<td>17</td>
</tr>
<tr>
<td>1983</td>
<td>Party songs</td>
<td>11</td>
</tr>
<tr>
<td>1966</td>
<td>Love</td>
<td>11</td>
</tr>
<tr>
<td>1964</td>
<td>Love</td>
<td>11</td>
</tr>
<tr>
<td>1976</td>
<td>Party songs</td>
<td>9</td>
</tr>
</tbody>
</table>

only showing top 5 rows
Cross Tabulation

```
scala> songs.stat.crosstab("YEAR", "THEME").show(5)
```

```
+-----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+
| YEAR_THEME      | Heartbreak     | Life and death | Love           | Party songs    | People and places | Politics and protest | Sex   |
+-----------------+----------------+----------------+----------------+----------------+----------------+----------------+-------+
| 1965            | 4              | 4              | 4              | 3              | 8              | 6              | 4      |
| 1928            | 0              | 2              | 1              | 0              | 2              | 0              | 1      |
| 2003            | 4              | 2              | 2              | 2              | 0              | 1              | 3      |
| 1950            | 0              | 1              | 0              | 0              | 1              | 0              | 0      |
| 1983            | 3              | 1              | 0              | 11             | 5              | 1              | 4      |
```

only showing top 5 rows
# WordCount using Spark DataFrames

```scala
# Read a RDD from file
val lines = sc.textFile("/home/ubuntu/13370.txt.utf-8")

# Parse RDD using FlatMap and create a DataFrame
val words = lines.flatMap(_.split(" ")).toDF("word")

# Group by Word and apply count function
val wordCountDF = words.groupBy("word").count()

# Print out the results
wordCountDF.show()
```

```
+---------+-----+
| word    | count|
|---------+-----+
| online  | 4    |
| By      | 9    |
| Text-Book | 1  |
| hope    | 8    |
| some    | 75   |
+---------+-----+
```

only showing top 5 rows
Spark SQL

• Supports most of the SQL standard
• SQL statements are compiled into Spark code and executed in cluster
• Can be used interchangeably with other Spark interfaces and libraries.
  – Spark Java, Scala, Python, R
  – DataFrames, MLlib
• Very similar to Hive, which uses MapReduce
  – But can avoid constantly having to define SQL schemas
  – Most user defined Hive functions are directly available
Spark SQL

• Provides the basic SQL like operations, such as:
  – Ability to filter rows from a table using a where clause.
  – Ability to select certain columns from the table using a select clause.
  – Ability to do equi-joins between two tables.
  – Ability to evaluate aggregations on multiple "group by" columns for the data stored in a table.
  – Ability to store the results of a query into another table.
  – Ability to download the contents of a table to a local directory.
  – Ability to store the results of a query in a hadoop dfs directory.
  – Ability to manage tables and partitions (create, drop and alter).
  – Ability to use custom scripts in chosen language as UDF’s.
# Load in data
val df = spark.read.json("/mnt/people.json")

#Register dataframe as SQL table
df.createOrReplaceTempView("people")

#Execute SQL query to create a new DataFrame
#Result is a DataFrame!
val adults = spark.sql("SELECT * FROM people WHERE age >= 18")

#Register the new dataframe also as a SQL table
adults.createOrReplaceTempView("adults")

#Show the results
adults.show()
Advanced SQL I

scala> val grouped = spark.sql("SELECT YEAR, THEME, COUNT(*) as count FROM songs GROUP BY THEME, YEAR")
scala> grouped.show(5)

+----------+-----------------+---+
| YEAR     | THEME           | count |
+----------+-----------------+---+
| 1979     | Party songs     | 17 |
| 1983     | Party songs     | 11 |
| 1964     | Love            | 11 |
| 1966     | Love            | 11 |
| 1976     | Party songs     | 9  |
+----------+-----------------+---+
```
scala> spark.sql("SELECT YEAR, THEME, count, 
SUM(count) over (PARTITION by THEME) as theme_count, 
SUM(count) over (PARTITION by YEAR) as year_count 
FROM grouped").show(100)
```

<table>
<thead>
<tr>
<th>YEAR</th>
<th>THEME</th>
<th>count</th>
<th>theme_count</th>
<th>year_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1987</td>
<td>Politics and protest</td>
<td>2</td>
<td>141</td>
<td>18</td>
</tr>
<tr>
<td>1970</td>
<td>Politics and protest</td>
<td>8</td>
<td>141</td>
<td>23</td>
</tr>
<tr>
<td>1987</td>
<td>Love</td>
<td>2</td>
<td>139</td>
<td>18</td>
</tr>
<tr>
<td>1970</td>
<td>Love</td>
<td>3</td>
<td>139</td>
<td>23</td>
</tr>
<tr>
<td>1957</td>
<td>Life and death</td>
<td>1</td>
<td>131</td>
<td>3</td>
</tr>
<tr>
<td>1941</td>
<td>Life and death</td>
<td>1</td>
<td>131</td>
<td>2</td>
</tr>
</tbody>
</table>
```
Extensions to SQL

- **collect_list**(col) - Returns a list of objects with duplicates.
- **ngrams**(sentences, int N, int K, int pf) - Returns the top-k N-grams from a set of tokenized sentences.
- **sentences**(string str, string lang, string locale) - Tokenizes a string of natural language text into words and sentences.
- **corr**(col1, col2) - Returns the Pearson coefficient of correlation of a pair of a numeric columns.
- **histogram_numeric**(col, b) - Computes a histogram of a numeric column in the group using b non-uniformly spaced bins.
### Histogram example

```sql
spark.sql("SELECT THEME, histogram_numeric(cast(YEAR as int),4) FROM songs GROUP BY THEME").show(5,false)
```

<table>
<thead>
<tr>
<th>THEME</th>
<th>histogram_numeric( CAST(YEAR AS INT), 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Politics and protest</td>
<td>[[1935.16,6.0], [1969.34,61.0], [1984.4,52.0], [2004.6,22.0]]</td>
</tr>
<tr>
<td>Love</td>
<td>[[1933.5,2.0], [1964.4,73.0], [1978.9,48.0], [1999.8125,16.0]]</td>
</tr>
<tr>
<td>Life and death</td>
<td>[[1933.0,4.0], [1949.5,2.0], [1968.6,63.0], [1993.3387096774186,62.0]]</td>
</tr>
<tr>
<td>People and places</td>
<td>[[1933.7,12.0], [1963.8783783783779,74.0], [1979.8,44.0], [2000.9,15.0]]</td>
</tr>
<tr>
<td>Heartbreak</td>
<td>[[1931.0,1.0], [1951.0,1.0], [1972.6,98.0], [1999.6,45.0]]</td>
</tr>
</tbody>
</table>

only showing top 5 rows
SparkR

- When using R, it is possible to import SparkR API and convert normal R DataFrame into a Spark R DataFrame.
- Spark R DataFrame implements
  - R dataframe operations
  - Spark DataFrame operations (SQL, MLlib, ...)
- Usable from R Studio and Shiny
Spark R Example

```r
library(SparkR, lib.loc = c(file.path(Sys.getenv("SPARK_HOME"), "R", "lib")))

sparkR.session(master = "local[*]", sparkConfig = list(spark.driver.memory = "2g"))

# Create a SparkDataFrame based using the faithful dataset from R.
df <- as.DataFrame(faithful)

# We use the `n` operator to count the number of times each waiting time appears
head(summarize(groupBy(df, df$waiting), count = n(df$waiting)))

  waiting count
  1     70    4
  2     67    1
  3     69    2

# Sort the output from the aggregation to get the most common waiting times
waiting_counts <- summarize(groupBy(df, df$waiting), count = n(df$waiting))

head(arrange(waiting_counts, desc(waiting_counts$count)))

  waiting count
  1     78   15
  2     83   14
  3     81   13
```
Spark Machine learning library - MLlib

• Implement and scale practical machine learning algorithms using Spark

• Provides:
  – Machine learning methods
  – Featurization (feature extraction, transformation, dimensionality reduction, and selection)
  – Pipelines
  – Persistence for Models and Pipelines
  – Utilities (linear algebra, statistics, data formats, UDF’s, etc.)
Machine Learning Methods

• **Classification & Regression**
  – Logistic regression
  – Decision tree classifier
  – Random forest classifier
  – Gradient-boosted tree classifier
  – Multilayer perceptron classifier
  – One-vs-Rest classifier (a.k.a. One-vs-All)
  – Naive Bayes

• **Clustering**
  – K-means
  – Latent Dirichlet allocation (LDA)
  – Bisecting k-means
  – Gaussian Mixture Model (GMM)

• **Collaborative filtering**
  – Recommender systems
MLlib Example

```scala
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors

def iris = spark.read.format("csv").load("/mnt/iris.data")

val dataset = iris.rdd.map(s =>
  Vectors.dense(s.getString(0).toDouble, s.getString(1).toDouble,
  s.getString(2).toDouble, s.getString(3).toDouble))

val clusters = KMeans.train(dataset, 3, 12)
val WSSSE = clusters.computeCost(dataset)

clusters.clusterCenters.foreach(println)
```
# Hadoop vs Spark

<table>
<thead>
<tr>
<th>Feature</th>
<th>Hadoop (MapReduce)</th>
<th>Spark (RDD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-memory computing</td>
<td>Not supported, uses HDD’s or HDFS.</td>
<td>Supported</td>
</tr>
<tr>
<td>Scalability</td>
<td>Very High</td>
<td>High</td>
</tr>
<tr>
<td>Can SQL be used to analyse data?</td>
<td>Yes (Hive, Impala)</td>
<td>Yes. Built into Spark.</td>
</tr>
<tr>
<td>Extensibility</td>
<td>High. Many existing UDF’s in Pig And Hive.</td>
<td>Very High. Includes Mllib. And uses the available UDF’s from Hive.</td>
</tr>
<tr>
<td>Streaming</td>
<td>Not supported directly.</td>
<td>Spark supports Streaming.</td>
</tr>
<tr>
<td>Purpose</td>
<td>Big Data &amp; simple tasks</td>
<td>Large scale data &amp; complex tasks</td>
</tr>
</tbody>
</table>
Thats All

• Next week`s practice session i
  – Processing data with Spark DataFrames
• Next week`s lecture is about data storage and querrying in NoSQL databases