Branch: master

Find file Copy path

CS167 / Labs / Lab6 / CS167_Lab6.md

tvu032 Update section number - Lab 6

e246727 on May 8

2 contributors

Raw Blame History

429 lines (387 sloc) 19.5 KB

Lab 6

Objectives

- Understand SparkSQL and the DataFrame API
- Write a Scala program that uses SparkSQL

Prerequisites

- Setup the development environment as explained in Lab 1.
- Download Apache Spark 2.4.5. Choose the package type *Pre-built for Apache Hadoop 2.7*.
- Download these two sample files sample file 1, sample file 2. Decompress the second file after download. These are the same files we used in Lab 3.
- Depending on how you extract the second file, it could be named either `nasa_19950630.22-19950728.12.tsv` or `19950630.23-19950801.00.tsv` . In this lab, we will use these two names interchangeably.
- For Windows users, install the Ubuntu app from Microsoft Store and set it up.
- To add Scala language support to IntelliJ, you can install the Scala plugin. Please check the plugin management page to see the details about installing and managing plugins in Intellij.

Lab Work

I. Project Setup (10 minutes)

1. Setup a new Scala project similar to Lab 5. Make sure to change the project name for Lab 6.
2. In the `pom.xml` file, add the following dependency.

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.12</artifactId>
  <version>2.4.5</version>
</dependency>
```

II. Initialize a SparkSession (5 minutes)

In this part, you will initialize your project with SparkSession to access SparkSQL and the DataFrame API.

1. In `App` class, add the following stub code.

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.SparkConf

object App {

  def main(args : Array[String]) {
    val conf = new SparkConf
    if (!conf.contains("spark.master"))
      conf.setMaster("local[*]")
    println(s"Using Spark master '${conf.get("spark.master")}'")

    val spark = SparkSession
      .builder()
      .appName("CS167 Lab6")
      .config(conf)
      .getOrCreate()

    try {
    } finally {
      spark.stop
    }
  }
}
```

III. Read and parse the input file (10 minutes)

Spark SQL is equipped with a CSV parser that can read semi-structured CSV files.

1. Use the following code to open the sample file and print the first few lines.

```
val input = spark.read.format("csv")
  .option("sep", "\t")
  .option("inferSchema", "true")
  .option("header", "true")
  .load("nasa_19950801.tsv")

import spark.implicits._

input.show()
```

The output should look similar to the following:

```
+--------------------+-------+---------+------+--------------------+--------+------+-------+---------+
|                host|logname|     time|method|                 url|response| bytes|referer|useragent|
+--------------------+-------+---------+------+--------------------+--------+------+-------+---------+
|pppa006.compuserv...|      -|807256800|   GET|/images/launch-lo...|     200|  1713|   null|     null|
|  vcc7.langara.bc.ca|      -|807256804|   GET|/shuttle/missions...|     200|  8677|   null|     null|
|pppa006.compuserv...|      -|807256806|   GET|/history/apollo/i...|     200|  1173|   null|     null|
|thing1.cchem.berk...|      -|807256870|   GET|/shuttle/missions...|     200|  4705|   null|     null|
|       202.236.34.35|      -|807256881|   GET|     /whats-new.html|     200| 18936|   null|     null|
|bettong.client.uq...|      -|807256884|   GET|/history/skylab/s...|     200|  1687|   null|     null|
|       202.236.34.35|      -|807256884|   GET|/images/whatsnew.gif|     200|   651|   null|     null|
|       202.236.34.35|      -|807256885|   GET|/images/KSC-logos...|     200|  1204|   null|     null|
|bettong.client.uq...|      -|807256900|   GET|/history/skylab/s...|     304|     0|   null|     null|
|bettong.client.uq...|      -|807256913|   GET|/images/ksclogosm...|     304|     0|   null|     null|
|bettong.client.uq...|      -|807256913|   GET|/history/apollo/i...|     200|  3047|   null|     null|
|        hella.stm.it|      -|807256914|   GET|/shuttle/missions...|     200|513911|   null|     null|
|mtv-pm0-ip4.halcy...|      -|807256916|   GET| /shuttle/countdown/|     200|  4324|   null|     null|
|   ednet1.osl.or.gov|      -|807256924|   GET|                   /|     200|  7280|   null|     null|
|mtv-pm0-ip4.halcy...|      -|807256942|   GET|/shuttle/countdow...|     200| 46573|   null|     null|
|dd10-046.compuser...|      -|807256943|   GET|/shuttle/missions...|     200| 10566|   null|     null|
|ad11-013.compuser...|      -|807256944|   GET|/history/history....|     200|  1602|   null|     null|
|dd10-046.compuser...|      -|807256946|   GET|/shuttle/missions...|     200|  8083|   null|     null|
|dd10-046.compuser...|      -|807256954|   GET|/images/KSC-logos...|     200|  1204|   null|     null|
|dd10-046.compuser...|      -|807256954|   GET|/history/apollo/i...|     200|  1173|   null|     null|
```

```
+--------------------+-------+---------+------+--------------------+--------+------+-------+---------+
only showing top 20 rows
```

2. The `option("sep", "\t")` configures the reader with the tab separator so there is no need to manually split each line.
3. The `option("header", "true")` will do two things. First, it will skip the first line in the file so that you do not have to manually remove it. Second, it will use the column names in that line to access the attribute so you do not need to access them by number.
4. The `option("inferSchema", "true")` tells Spark to infer the schema based on the values in the file. For example, the time will be parsed as an integer number which allows the use of range comparison.
5. To check the inferred schema, add the following statement.

```
input.printSchema()
```

The output should look similar to the following:

```
root
 |-- host: string (nullable = true)
 |-- logname: string (nullable = true)
 |-- time: integer (nullable = true)
 |-- method: string (nullable = true)
 |-- url: string (nullable = true)
 |-- response: integer (nullable = true)
 |-- bytes: integer (nullable = true)
 |-- referer: string (nullable = true)
 |-- useragent: string (nullable = true)
```

6. Comment the line `option("inferSchema", "true")` and run your program again. (Q) What is the type of the attributes `time` and `bytes` this time? Why?

IV. Query the Dataframe using Dataframe Operators (60 minutes)

In this part, we will run some relational operators through the Dataframe API. The logic of these queries is similar to what we did in Lab 5. This will allow you to compare and contrast the two APIs.

1. Add the following code (similar to Lab 5) to run a user-provided operation.

```
val command: String = args(0)
val inputfile: String = args(1)

...

val t1 = System.nanoTime
command match {
  case "count-all" =>

  // TODO count total number of records in the file
  case "code-filter" =>
  // TODO Filter the file by response code, args(2), and print the total number of matching lines
  case "time-filter" =>
  // TODO Filter by time range [from = args(2), to = args(3)], and print the total number of matching lines
  case "count-by-code" =>
  // TODO Group the lines by response code and count the number of records per group
  case "sum-bytes-by-code" =>
  // TODO Group the lines by response code and sum the total bytes per group
  case "avg-bytes-by-code" =>
  // TODO Group the liens by response code and calculate the average bytes per group
  case "top-host" =>
  // TODO print the host the largest number of lines and print the number of lines
  case "comparison" =>
  // TODO Given a specific time, calculate the number of lines per response code for the
  // entries that happened before that time, and once more for the lines that happened at or after
  // that time. Print them side-by-side in a tabular form.
```

```
}
val t2 = System.nanoTime
println(s"Command '${command}' on file '${inputfile}' finished in ${(t2-t1)*1E-9} seconds")
```

2. The operation `count-all` is implemented using the `count` function. The output should look similar to the following.

```
Total count for file 'nasa_19950801.tsv' is 30969
Total count for file '19950630.23-19950801.00.tsv' is 1891709
```

3. The operation `code-filter` should count the records with a give response code. To do that, you will use the `filter` method. The easiest way is to provide the test as a string, e.g., `"response=200"`. Alternatively, you can use the expression `$"response" === 200`. For the latter, make use that you ipmort the implicit coversion using the statement `import spark.implicits._` in your program. The output should look similar to the following.

```
Total count for file 'nasa_19950801.tsv' with response code 200 is 27972
Total count for file '19950630.23-19950801.00.tsv' with response code 302 is 46573
```

4. The operation `time-filter` should count all the records that happened in a time interval `[start, end]`. You will use the `filter` function but this time with the `between` expression. Again, you can just provide the filter predicate as a string, i.e., `"time BETWEEN 807274014 AND 807283738"`, or as a Scala expression, i.e., `$"time".between(807274014, 807283738)`. This will be followed by `count` to count the records. A sample output is given below.

```
Total count for file 'nasa_19950801.tsv' in time range [807274014, 807283738] is 6389
Total count for file '19950630.23-19950801.00.tsv' in time range [804955673, 805590159] is 554919
```

5. The two operations `count-by-code`, `sum-bytes-by-code`, and `avg-bytes-by-code` will all look very similar. You first need to group records by response code using the `groupBy` function, i.e., `groupBy("response")` or `groupBy($"response")`. On the result, you should call the correct aggregate function, i.e., `count`, `sum`, or `avg`. The last two functions take a parameter which is the column name to aggregate, e.g., `sum("bytes")`. You can finally print the result using the `show()` command. The output should look like the following.

```
Number of lines per code for the file nasa_19950801.tsv
+--------+-----+
|response|count|
+--------+-----+
|     404|  221|
|     200|27972|
|     304| 2421|
|     302|  355|
+--------+-----+



Number of lines per code for the file 19950630.23-19950801.00.tsv
+--------+-------+
|response|  count|
+--------+-------+
|     501|     14|
|     500|     62|
|     403|     54|
|     404|  10845|
|     200|1701534|
|     304| 132627|
|     302|  46573|
+--------+-------+



Total bytes per code for the file nasa_19950801.tsv
+--------+----------+
|response|sum(bytes)|
```

```
+--------+----------+
|     404|         0|
|     200| 481974462|
|     304|         0|
|     302|     26005|
+--------+----------+


Total bytes per code for the file 19950630.23-19950801.00.tsv
+--------+-----------+
|response| sum(bytes)|
+--------+-----------+
|     501|          0|
|     500|          0|
|     403|          0|
|     404|          0|
|     200|38692291442|
|     304|          0|
|     302|    3682049|
+--------+-----------+

Average bytes per code for the file nasa_19950801.tsv
+--------+------------------+
|response|        avg(bytes)|
+--------+------------------+
|     404|               0.0|
|     200|17230.604247104246|
|     304|               0.0|
|     302| 73.25352112676056|
+--------+------------------+



Average bytes per code for the file 19950630.23-19950801.00.tsv
+--------+------------------+
|response|        avg(bytes)|
+--------+------------------+
|     501|               0.0|
|     500|               0.0|
|     403|               0.0|
|     404|               0.0|
|     200|22739.652244386536|
|     304|               0.0|
|     302|  79.0597341807485|
+--------+------------------+
```

6. The operation `top-host` should group records by host, `groupBy("host")`, then count records in each group `count()`. After that, you should sort the results in descending order by count, `orderBy($"count".desc)`. Finally, return the top result using the method `first()`. The final result will be of type `Row`. To access the host and number of records for the top result, you can use one of the methods `Row#getAs(String)` and `Row#getAs(Int)` which retrieve an attribute by its name and index, respectively. The final output should look similar to the following.

```
Top host in the file nasa_19950801.tsv by number of entries
Host: edams.ksc.nasa.gov
Number of entries: 364



Top host in the file 19950630.23-19950801.00.tsv by number of entries
Host: piweba3y.prodigy.com
Number of entries: 17572
```

7. Finally, the operation `comparison` should count records by response code before and after a specific timestamp. You can do that by first creating two Dataframe by filtering the input twice. For each Dataframe, you can count the records by response code as done in the operation `count-by-code`. Finally, you can join the results of the two Dataframes by code to place them side-by-side in one Dataset. The join method may look like the following line:

```
countsBefore.join(countsAfter, "response")
```

which joins two dataframes, namely, `countsBefore` and `countsAfter`, using the common key `response`. You can then print out the final result using the `show` command as follows.

```
Comparison of the number of lines per code before and after 807295758 on file nasa_19950801.tsv
+--------+------------+-----------+
|response|count_before|count_after|
+--------+------------+-----------+
|     404|         199|         22|
|     200|       22248|       5724|
|     304|        1925|        496|
|     302|         272|         83|
+--------+------------+-----------+


Comparison of the number of lines per code before and after 805383872 on file 19950630.23-19950801.00.tsv
+--------+------------+-----------+
|response|count_before|count_after|
+--------+------------+-----------+
|     501|           2|         12|
|     500|          53|          9|
|     403|          19|         35|
|     404|        3864|       6981|
|     200|      594412|    1107122|
|     304|       38000|      94627|
|     302|       21057|      25516|
+--------+------------+-----------+
```

Hint: By default, the name of the column that results from the `count` function is named `count`. You can rename this column in each Datafame separately using the method `withColumnRenamed`, for example, for the count-before dataframe, the stament will be `withColumnRenamed("count", "count_before")`.

V. Submission (15 minutes)

1. Add a `README` file with all your answers.
2. If you implemented the bonus task, add your explanation and code snippet to the `README` file.
3. Add a `run` script that compiles your code and then runs the following commands with the given parameters on the file `nasa_19950630.22-19950728.12.tsv`.

Command	Parameters
count-all	
code-filter	302
time-filter	804955673 805590159
count-by-code	
sum-bytes-by-code	
avg-bytes-by-code	
top-host	
comparison	805383872

4. As a test, run your script using the following command to redirect the standard output to the file `output.txt` and double check that the answers in your file are the same to the ones listed earlier in this lab for the file `nasa_19950630.22-19950728.12.tsv`.

```
./run.sh > output.txt
```

Further Readings

The folllowing reading material could help you with your lab.

- Spark SQL Programming Guide
- Dataset API Docs

FAQ

- Q: My code does not compile using `mvn package`.
- Q: IntelliJ IDEA does not show the green run arrow next to the `App` class.
- A: Check your `pom.xml` file and make sure that the following sections are there in your file.

```
<properties>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
  <encoding>UTF-8</encoding>
  <scala.version>2.12.6</scala.version>
  <scala.compat.version>2.12</scala.compat.version>
  <spec2.version>4.2.0</spec2.version>
  <spark.version>2.4.5</spark.version>
</properties>


<dependencies>
  <dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>${scala.version}</version>
  </dependency>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.12</artifactId>
    <version>${spark.version}</version>
    <scope>compile</scope>
  </dependency>

  <!-- Test -->
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.scalatest</groupId>
    <artifactId>scalatest_${scala.compat.version}</artifactId>
    <version>3.0.5</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.specs2</groupId>
    <artifactId>specs2-core_${scala.compat.version}</artifactId>
    <version>${spec2.version}</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.specs2</groupId>
    <artifactId>specs2-junit_${scala.compat.version}</artifactId>
    <version>${spec2.version}</version>
    <scope>test</scope>
```

```
    </dependency>
  </dependencies>

  <build>
    <sourceDirectory>src/main/scala</sourceDirectory>
    <testSourceDirectory>src/test/scala</testSourceDirectory>
    <plugins>
      <plugin>
        <!-- see http://davidb.github.com/scala-maven-plugin -->
        <groupId>net.alchim31.maven</groupId>
        <artifactId>scala-maven-plugin</artifactId>
        <version>3.3.2</version>
        <executions>
          <execution>
            <goals>
              <goal>compile</goal>
              <goal>testCompile</goal>
            </goals>
            <configuration>
              <args>
                <arg>-dependencyfile</arg>
                <arg>${project.build.directory}/.scala_dependencies</arg>
              </args>
            </configuration>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.21.0</version>
        <configuration>
          <!-- Tests will be run with scalatest-maven-plugin instead -->
          <skipTests>true</skipTests>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.scalatest</groupId>
        <artifactId>scalatest-maven-plugin</artifactId>
        <version>2.0.0</version>
        <configuration>
          <reportsDirectory>${project.build.directory}/surefire-reports</reportsDirectory>
          <junitxml>.</junitxml>
          <filereports>TestSuiteReport.txt</filereports>
          <!-- Comma separated list of JUnit test class names to execute -->
          <jUnitClasses>samples.AppTest</jUnitClasses>
        </configuration>
        <executions>
          <execution>
            <id>test</id>
            <goals>
              <goal>test</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
```