Lab 5

Objectives

- Use Scala API to access Spark RDD.
- Get familiar with the various transformations and actions in Spark.

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.
- (Optional) 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.

Overview

In this lab, we will be using mainly Scala code. While pure Scala projects are usually set up using SBT, we will use Maven for this project to reuse your existing development environment and avoid the complications of setting up a new development tool.

Lab Work

I. Project Setup (20 minutes)

We will follow the instructions on the official Scala website.

1. Run the following command:

```
mvn archetype:generate
```

2. Maven will generate a long list of all artifacts. It will then prompt for the artifact ID number or a search text to narrow down the list. The prompt will look similar to the following:

```
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): 1591:
```

3. To narrow down the list, type `net.alchim31.maven:scala-archetype-simple` and hit enter.
4. You should only see one item with ID 1 similar to the following:

```
Choose archetype:
1: remote -> net.alchim31.maven:scala-archetype-simple (The maven-scala-plugin is used for
compiling/testing/running/documenting scala code in maven.)
```

5. Type 1 and hit enter to choose that artifact.
6. It will prompt for the version, choose the latest one by entering its number. The prompt will look similar to the following. Just hit enter to accept the default choice which is 4 in this case.

```
Choose net.alchim31.maven:scala-archetype-simple version:
1: 1.4
2: 1.5
3: 1.6
4: 1.7
Choose a number: 4:
```

7. Next, it will ask for groupId, enter `edu.ucr.cs.cs167.<UCRNetID>` . *(Use your UCRNetID)*
8. Next, it will ask for the artifactId, enter `<UCRNetID>_lab5` . *(Use your UCRNetID)*
9. Hit enter twice to accept the default version and package.
10. Review the configuration and enter `Y` to confirm.
11. Change into the project directory and type `mvn package` once to make sure that it compiles.
12. To configure your project with Spark, merge the following configuration in `pom.xml` .

```
<properties>
  <spark.version>2.4.5</spark.version>
</properties>

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

13. Import your project into IntelliJ IDEA in the same way you did in previous labs and make sure it compiles. Run the main function in `App` class to make sure it works.
14. To make it easier to run your code from the JAR file, add the following part to the plugins section in your `pom.xml` file.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <configuration>
    <archive>
      <manifest>
        <mainClass>edu.ucr.cs.cs167.[UCRNetID].App</mainClass>
      </manifest>
    </archive>
  </configuration>
</plugin>
```

II. Initialize with Spark (5 minutes)

In this part, you will initialize your project with Spark.

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

```
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}

object App {

  def main(args : Array[String]) {
    val command: String = args(0)
    val inputfile: String = args(1)

    val conf = new SparkConf
    if (!conf.contains("spark.master"))
      conf.setMaster("local[*]")
    println(s"Using Spark master '${conf.get("spark.master")}'")
    conf.setAppName("lab5")
    val sparkContext = new SparkContext(conf)
    try {
      val inputRDD: RDD[String] = sparkContext.textFile(inputfile)
      // TODO Parse the input file using the tab separator and skip the first line
      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")
    } finally {
      sparkContext.stop
    }
  }
}
```

2. Take a few minutes to check the stub code and understand what it does. It has to required command-line arguments. (Q) What are these two arguments?

III. Read and parse the input file (15 minutes)

1. Since most of the commands will need to split the input line and skip the first line, let us do this first.
2. Use a filter transformation to skip the first line. For simplicity, we will detect the first line as the line that starts with `"host\tlogname"`
3. Use the map transformation to split each line using the tab character `"\t"` as a separator.
4. Note that since the filter and map operations are transformations, not action, none of them will be executed until you use them.

5. If you get the error `java.lang.NoClassDefFoundError: scala/runtime/LambdaDeserialize`, go to `pom.xml` and change scala version to `2.11.11`.

```
<scala.version>2.11.11</scala.version>
<scala.compat.version>2.11</scala.compat.version>
```

You should change Spark version as well:

```
<artifactId>spark-core_2.11</artifactId>
```

6. Few commands in the next sections may require more than 2 arguments.

IV. `count-all` and `code-filter` (15 minutes)

1. The `count-all` command should use the method `RDD#count` which is an action. Below is the expected output for the two sample files.

```
Total count for file 'nasa_19950801.tsv' is 30970
Total count for file '19950630.23-19950801.00.tsv' is 1891710
```

Note that if you count the lines after filtering the header, the numbers will be 1 less.

2. The `code-filter` command should count the lines that match a desired response code. The desired code is provided as a third command line argument. This method should use the `filter` transformation followed by the `count` action. Below is the expected output for the two sample files.

```
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
```

- Note: For all commands in this lab, make sure that you write the output to the standard output using the `print` command and that the output looks exactly the same to the expected output. We will use a script to automatically check your answer and it will use regular expressions to match the answer. Any change in this expected output might reduce your grade for this lab.
- Hint: To make your code more readable, you can add constants for each attribute to access them by name instead of number. See the following code snippet for an example.

```
val ResponseCode: Int = 5
val code: String = line.split("\\t")(ResponseCode)
```

V. `time-filter` (10 minutes)

1. In this part, we need to count the number of lines that have a timestmap in a given range `[start, end]`.
2. The interval is given as two additional arguments as integers.
3. Do not forget to use the method `String#toLong` in Scala to convert the String argument to a long integer.
4. Similar to `code-filter`, you will need a filter followed by count to complete this part.
5. 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
```

VI. `count-by-code` (15 minutess)

1. This part requires grouping the records by response code first. In Scala, this is done using a map operation that returns a tuple `(key,value)`.
2. You can directly count each group using the function `countByKey`.
3. To print the output on the resulting map, you can use the method `foreach` on that map. A sample output is given below.

```
Number of lines per code for the file nasa_19950801.tsv
Code,Count
404,221
200,27972
302,355
304,2421
```

```
Number of lines per code for the file 19950630.23-19950801.00.tsv
Code,Count
302,46573
501,14
404,10845
500,62
403,54
304,132627
200,1701534
```

VII. `sum-bytes-by-code` and `avg-bytes-by-code` (15 minutes)

1. This method is similar to the previous one except that it will calculate the summation of bytes for each code.
2. To do that, you can first use the `map` function to produce only the `code` and the `bytes`. Then, you can use the mehod `reducyByKey` to compute the summation.
3. The reduce method is Spark is different that the reduce method in Hadoop. Instead of taking all the values, it only takes two values at a time. To compute the summation, your reduce function should return the sum of the two values given to it.
4. Since reduceByKey is a transformation, you will need to use the `collect` action to get the results back.
5. A sample output is given below.

```
Total bytes per code for the file nasa_19950801.tsv
Code,Sum(bytes)
404,0
200,481974462
302,26005
304,0
```

```
Total bytes per code for the file 19950630.23-19950801.00.tsv
Code,Sum(bytes)
501,0
403,0
304,0
200,38692291442
404,0
302,3682049
500,0
```

6. For `avg-bytes-by-code` you need to compute the average, rather than the summation. A simple reduce function cannot be used to compute the average since the average function is not associative. However, it can be computed using a combination of sum and count.

7. The easiest way to compute the average is to combine the output of the two commands `count-by-code` and `sum-bytes-by-code`. The average is simply the sum divided by count.
8. Bonus (+3 points): The drawback of this method is that it will need to scan the input twice to count each function, sum and count. It is possible to compute both functions in one scan over the input and without caching any intermediate RDDs. Complete this part to get three bonus points on this lab. Explain your method in the README file and add the code snippet that performs this task. Mark your answer with (B).
9. A sample output is given below.

```
Average bytes per code for the file nasa_19950801.tsv
Code,Avg(bytes)
404,0.0
200,17230.604247104246
302,73.25352112676056
304,0.0



Average bytes per code for the file 19950630.23-19950801.00.tsv
Code,Avg(bytes)
501,0.0
403,0.0
304,0.0
200,22739.652244386536
404,0.0
302,79.0597341807485
500,0.0
```

VIII. `top-host` (20 minutes)

1. In this part we want to count the number of entries per host and output the one with the highest number of entries.
2. While we could use the function `countByKey` it could be inefficient since it returns all the values to the driver node. Unlike the response codes, there could be too many distance values of `host` and we do not want to return all of them.
3. Instead of `countByKey` we will use the method `reduceByKey` which runs as a transformation and keeps the result in an RDD.
4. After that, we will use the transformation `sortBy` to sort the results in *descending* order.
5. Finally, we will use the action `first` to return only the first value.
6. Sample output

```
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
```

IX. `comparison` (10 minutes)

In this part, we would like to split the input into two parts based on a timestamp and then calculate the number of lines for each response code for each part.

1. To split the input, we will use two filter transformations. Each one will result in a different RDD.
2. After that, we will reuse our code from the command `count-by-code` to count the number of records per response code.
3. A sample output is given below.

```
Comparison of the number of lines per code before and after 807295758 on file nasa_19950801.tsv
Code,Count before,Count after
404,199,22
200,22248,5724
302,272,83
304,1925,496
```

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

X. 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.

- RDD Programming Guide
- RDD 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>
```