ISIT312 Big Data Management

Session 4, 2021

Exercise 8

Spark Fundamentals

In this exercise, you will learn basic operations in Spark's Shell and development of simple Spark applications. This will help you gain proper understanding on Spark and prepare to work at the tasks included in the Assignments.

Be careful when copying the Linux commands in this document to your working Terminal, because it is error-prone. Maybe you should type those commands by yourself.

Prologue
Login to your system and start VirtualBox.

When ready start a virtual machine `ISIT312-BigDataVM-07-SEP-2020`.

(1) How to start Hadoop?

Open a new Terminal window and start Hadoop in the following way.

```
$HADOOP_HOME/sbin/start-all.sh
```

(2) Spark-shell quick start

In the Big Data VM, Spark-shell can be started in two modes: pseudo-distributed mode and local mode. The following command starts Spark-shell in the pseudo-distributed mode.

```
$SPARK_HOME/bin/spark-shell --master yarn
```

Besides integrating with Hadoop YARN, Spark comes with its own cluster manager, e.g. standalone cluster manager, see the lecture notes for more information.

The following command starts Spark-shell in the local mode.

```
$SPARK_HOME/bin/spark-shell --master local[*]
```

The `*` symbol means using multiple threads in the VM to process a Spark job. It is recommended you use the "local" master for efficiency reasons.

Spark-shell runs on top of the Scala REPL. To quit Scala REPL, type

```
:quit
```

When Spark-shell is started, a SparkSession instance named `spark`, which is the entry points to a Spark application. To view it, simply type:

```
spark
```

With this SparkSession instance, we can create DataFrames.

```
val myRange0 = spark.range(20).toDF("number")
myRange0.show()
val myRange1 = spark.range(18).toDF("numbers")
myRange1.show()
myRange0.except(myRange1).show()
```

Open another Terminal window and upload the file `README.md` located in `$SPARK_HOME` to HDFS.

```
cd $SPARK_HOME
$HADOOP_HOME/bin/hadoop fs -put README.md README.md
```

Next, read it into a DataFrame.

```
val YOUR_HDFS_PATH="."
val textFile = spark.read.textFile("./README.md")
```

Next, count the total number of lines and display the first line in the file.

```
textFile.count()
textFile.first()
```

Next, count how many lines contain the word "Spark".

```
textFile.filter(line => line.contains("Spark")).count()
```

Next, find frequencies of each word in the document.

```
textFile.map(line => line.split(" ").size).reduce((a, b) => if (a > b) a else b)
val wordCounts = textFile.flatMap(line => line.split(" ")).groupByKey(identity).count()
wordCounts.show()
wordCounts.collect()
```

(3) Run Scala script in Spark-shell

Process a command

```
:quit
```

to quit Spark-shell.

Open a plain document in Text Editor (`gedit`).

Insert the following Scala commands into the document and save it as `myScalaScript.txt` in the working directory.

```
val YOUR_HDFS_PATH="."
val textFile = spark.read.textFile("./README.md")
textFile.count()
textFile.first()
textFile.filter(line => line.contains("Spark")).count()
textFile.map(line => line.split(" ").size).reduce((a, b) => if (a > b) a else b)
```

```
val wordCounts = textFile.flatMap(line => line.split(" ")).groupByKey(identity).count()
wordCounts.show()
wordCounts.collect()
```

Restart Spark-shell in the local mode (see step (2)).

Process script `myScalaScript.txt` in Spark-shell in the following way.

```
:paste myScalaScript.txt
```

Note, that only the results of `wordCounts.show()` and `wordCounts.collect()` are displayed.

(4) DataFrame/Dataset transformations and actions

Download the files `people.json`, `people.txt` and `employees.json` to your virtual machine.

Then, in Terminal window, upload the files to HDFS in the following way.

```
cd ~
$HADOOP_HOME/bin/hadoop fs -put people.json people.json
$HADOOP_HOME/bin/hadoop fs -put people.txt people.txt
$HADOOP_HOME/bin/hadoop fs -put employees.json employees.json
$HADOOP_HOME/bin/hadoop fs -ls
```

Next, display the contents of a file `people.json`.

```
$HADOOP_HOME/bin/hadoop fs -cat people.json
```

Type the following DataFrame/Dataset operations in Spark-shell window to read `people.json` file into a dataframe and to display its contents and structure.

```
val df = spark.read.json("./people.json")
df.show()
df.printSchema()
```

Next, perform few basic operations on the contents of dataframe.

```
df.select($"name", $"age" +1 ).show()
df.filter($"age" >21).show()
df.groupBy("age").count().show()
df.createOrReplaceTempView("people")
val sqlDF = spark.sql("select * from people")
sqlDF.show()
```

Does it remind you SQL ? Yes, it is "the same chicken but in a bit different gravy".

Next, create a Dataset in the following way.

```
case class Person(name: String, age: Long)
val ccDS = Seq(Person("Andy", 32)).toDS()
ccDS.show()
```

```
ccDS.select($"name").show()
```

Next, practice another way to create DataFrame. This time we shall use a text file `people.txt` already uploaded to HDFS.

```
val peopleDF =
spark.sparkContext.textFile("./people.txt").map(_.split(",")).map(
attributes=>Person(attributes(0),attributes(1).trim.toInt)).toDF()
```

And verify its contents with

```
peopleDF.show()
```

Next, we save DataFrame as DataSet.

```
case class Employee(name: String, salary: Long)
val ds = spark.read.json("./employees.json").as[Employee]
```

And verify it contents with

```
ds.show()
```

(5) Self-contained application

In the following, we implement a self-contained application and submit it as a Spark job.

Open a new document in Text Editor, input the following code and save it as a file `SimpleApp.scala`.

```
import org.apache.spark.sql.SparkSession
object SimpleApp
{
  def main(args: Array[String])
  {
    val text = "./README.md"
    val spark = SparkSession.builder.appName("Simple
Application").config("spark.master", "local[*]").getOrCreate()
    val data = spark.read.textFile(text).cache()
    val numAs = data.filter(line => line.contains("a")).count()
    val numBs = data.filter(line => line.contains("b")).count()
    println(s"Lines with a: $numAs, Lines with b: $numBs")
    spark.stop()
  }
}
```

Compile `SimpleApp.scala` by the following command in the Terminal.

```
scalac -classpath "$SPARK_HOME/jars/*" SimpleApp.scala
```

Then create a jar file in the following way.

```
jar cvf app.jar SimpleApp*.class
```

Finally, process it with Spark-shell in the following way.

```
$SPARK_HOME/bin/spark-submit --master local[*] --class SimpleApp
app.jar
```

When ready, retrieve the following line from a "jungle" of messages generated by Spark-shell.

```
Lines with a: 62, Lines with b: 30
```

(6) Shakespeare wordcount exercise

Complete the following exercise:
Objective: Count the frequent words used by William Shakespeare, but remove the known English stops words (such as “the”, “and” and “a”) in stop-words-list.csv. Return top 20 most frequent non-stop words in Shakespeare’s works.

Data sets: `shakespeare.txt`, `stop-words-list.csv`

For a good start the first few lines of code are provided below.

```
val shakes = spark.read.textFile(".../shakespeare.txt")
val swlist = spark.read.textFile(".../stop-word-list.csv")
val shakeswords = shakes.flatMap(x =>
x.split("\\W+")).map(_.toLowerCase.trim).filter(_.length>0)shake
swords.createOrReplaceTempView("shakeswords")
val stopwords = swlist.flatMap(x=>x.split(",")).map(_.trim)
stopwords.createOrReplaceTempView("stopwords")
```

The final output should be as follows:

```
result.show(20)
+-----+-----+
|value|count|
+-----+-----+
| d   | 8608|
| s   | 7264|
| thou| 5443|
| thy | 3812|
|shall| 3608|
| thee| 3104|
| o   | 3050|
| good| 2888|
| now | 2805|
| lord| 2747|
| come| 2567|
| sir | 2543|
| ll  | 2480|
| here| 2366|
| more| 2293|
| well| 2280|
| love| 2010|
| man | 1987|
| hath| 1917|
| know| 1763|
+-----+-----+
only showing top 20 rows
```

End of exercise 8