SIBD final project

Davide Peron
Cristina Gava

January 16, 2018

1. Solution's implementation

The first step to do was the execution of the Recommender in IntelliJ, using the Spark and Scala packets. To do it we firstly cleaned the dataset and took a subset of it in order to have a faster execution of the code. The dataset had to be cleaned from spurious characters which where present in artists' names and were not recognized as standard ASCII, and the code used for that is in the file *filter_artist_data.py* and is shown here for completeness.

```python
inputfile=open("artist_data.txt", "r")
outputfile = open("artist_data_filtered.txt", "w")
i = 0

for line in inputfile:
    line_array = line.split("\t")
    i += 1
    if(len(line_array) == 2):
        outputfile.write(line)
```

After that we took a subset of artist data, which now is 184 Bytes long: we are aware that this dimension corresponds only to a few lines from the dataset, but we did so in order to execute the recommender in a little amount of time, given the power of our laptops. Using decentralized web services like AWS we would have been able to run the entire dataset, however the results of the code can be appreciated also with our small subset.

The next step was to modify the Recommender file code to make it execute on our machine: as it can be seen from the code below, first of all we imported the *Source* library in order to deploy the args[] option and make the userID readable from command line. After that we declared the url for the hadoop server and port and made the code able to read the text files containing the dataset. Finally we added some *println* instructions to see the evolution of each subsection of the code.

```scala
import scala.io.Source

object RunRecommender {

  def main(args: Array[String]): Unit = {

    Logger.getLogger("org").setLevel(Level.OFF)
    Logger.getLogger("akka").setLevel(Level.OFF)

        val userID = args(0).toInt
    val spark = SparkSession.builder().master("local").getOrCreate()
    // Optional, but may help avoid errors due to long lineage
    // spark.sparkContext.setCheckpointDir("hdfs://localhost:54310/tmp/")

    //val base = "hdfs://localhost:54310/user/ds/"
```

```
      val base = "hdfs://hadoop:9000/user/ds/"
    val rawUserArtistData = spark.read.textFile(base + "user_artist_data.txt")
    val rawArtistData = spark.read.textFile(base + "artist_data_filtered.txt")
    val rawArtistAlias = spark.read.textFile(base + "artist_alias.txt")

    val runRecommender = new RunRecommender(spark)
    println("Preparation")
    runRecommender.preparation(rawUserArtistData, rawArtistData, rawArtistAlias)
    println("Model")
    runRecommender.model(rawUserArtistData, rawArtistData, rawArtistAlias,
        userID)
    println("Evaluation")
    runRecommender.evaluate(rawUserArtistData, rawArtistAlias)
    println("Recommendation")
    runRecommender.recommend(rawUserArtistData, rawArtistData, rawArtistAlias)
  }
```

2. SBT dependencies

The dependencies for the Recommender have been managed through the *build.sbt* file, here shown below.

```
name := "RecommenderSBT"

version := "0.1"

scalaVersion := "2.11.12"

val sparkVer = "2.1.0"

libraryDependencies ++= Seq(
  "org.apache.spark" %% "spark-core" % sparkVer,
  "org.apache.spark" %% "spark-sql" % sparkVer,
  "org.apache.spark" %% "spark-mllib" % sparkVer,
  "com.github.fommil.netlib" % "all" % "1.1.2" pomOnly()
)
```

3. HDFS File System using Hadoop

To correctly run the recommender we used the HDFS file system offered by Apache Hadoop. Here we uploaded the dataset we have to analyse and we wrote in the recommender code where to take the data from with the syntax *hdfs://host_name:port/path/of/dataset*.

The whole code then has been executed in the IntelliJ IDE and the recommender gave a few suggestions for the selected user.

4. Docker containers configuration

For the implementation in Docker we created a *Docker Compose* file in which we listed the two services for Hadoop and Spark. The code contained in the file is listed here:

```
version: '2'
services:
  spark:
    image: "p7hb/docker-spark:2.1.0"
    container_name: spark
    volumes:
      - ./shared:/shared
    ports:
      - 4040:4040
      - 8080:8080
      - 8081:8081
    command: bash -c "/shared/spark_setup.sh"
  hadoop:
    image: "sequenceiq/hadoop-docker:2.7.1"
    container_name: hadoop
    volumes:
      - ./shared:/shared
    ports:
      - 9000:9000
    command: bash -c "/etc/bootstrap.sh -bash && /shared/db_setup.sh"
```

As it can be seen from the code, we first declared the Spark service and the ports through which make the container available.

The first thing that Docker Compose does, if not already done, is to pull the images of both the containers used in the services. We also created a directory which is in common between the two containers, in order to upload the dataset only in one place and offer an environment in which the containers could share files without problems. The same settings have been modelled for Hadoop.

An important part in the yml file is the definition of the command files *spark_setup.sh* and *db_setup.sh*. During the first attempts with the code we incurred in some problems regarding the order of execution of the services: we noticed that sometimes the program was crashing due to the fact that Spark required the dataset before Hadoop was able to fully set the connection to it. So we wrote:

- **spark_setup.sh**

```
/bin/echo 'Welcome in Ziggy, the new generation of Recommender!!'
/bin/echo '

#I keep Spark sleeping until Hadoop is not ready
while [ ! -e /shared/ready.txt ]
do
```

```
    sleep 1
done

rm -rf /shared/ready.txt

#then I start the Recommender

/bin/echo 'Ziggy is running the Recommender...'
spark-submit --class com.cloudera.datascience.recommender.RunRecommender
    /shared/recommender_2.11-0.1.jar 2093760
```

and

- **db_setup.sh**

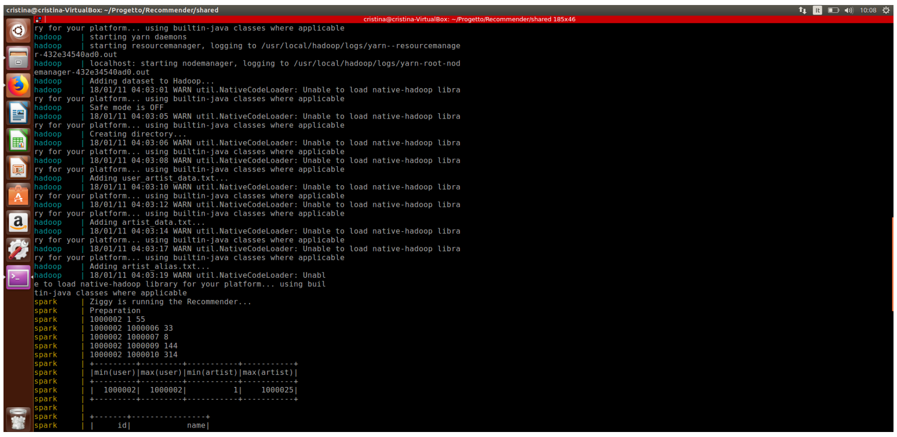
```
/bin/echo 'Adding dataset to Hadoop...'

#Adding Hadoop to PATH
PATH="$PATH:/usr/local/hadoop/bin"
hdfs dfsadmin -safemode leave

#Adding dataset to Hadoop checking that it doesn't exists yet

hadoop fs -test -e /user/ds
if [ $? != 0 ]
then
    /bin/echo reating directory...'
    hadoop fs -mkdir /user/ds
fi

hadoop fs -test -e /user/ds/user_artist_data.txt
if [ $? != 0 ]
then
    /bin/echo '<dding user_artist_data.txt...'
    hadoop fs -put /shared/user_artist_data.txt /user/ds/user_artist_data.txt
fi

hadoop fs -test -e /user/ds/artist_data_filtered.txt
if [ $? != 0 ]
then
    /bin/echo 'Adding artist_data.txt...'
    hadoop fs -put /shared/artist_data_filtered.txt
        /user/ds/artist_data_filtered.txt
fi

hadoop fs -test -e /user/ds/artist_alias.txt
if [ $? != 0 ]
then
    /bin/echo 'Adding artist_alias.txt...'
    hadoop fs -put /shared/artist_alias.txt /user/ds/artist_alias.txt
fi

echo "I'm ready" >> /shared/ready.txt
chmod 777 /shared/ready.txt

while true
do
    sleep 1
done
```

In the *spark_setup* we said to Spark to wait until Hadoop is ready (it will understand when this moment comes checking the existence of *ready.txt* file in the shared directory), at the same time the file *Docker Compose* starts the Hadoop service and the correspondent container. When the Hadoop container is ready, the file *db_setup* executes: the first thing it does is to check if the dataset is already added, if not so, it is uploaded. When every file is ready, Hadoop creates the file *ready.txt* setting its user permission to 777 mode (everyone has full access to the file) so as to be possible for the Spark service to read it and delete it after.

After the file creation, Hadoop enters a sort of *listening* mode, in which it waits for the Spark container to run the recommender, and finally produces an *end.txt* file. So, while Hadoop is waiting, Spark detects the existence of *ready.txt*, it deletes it and runs the recommender. Once the recommender finished its tasks, the Spark service produces the *end.txt* file with, again, 777 permissions and stops.

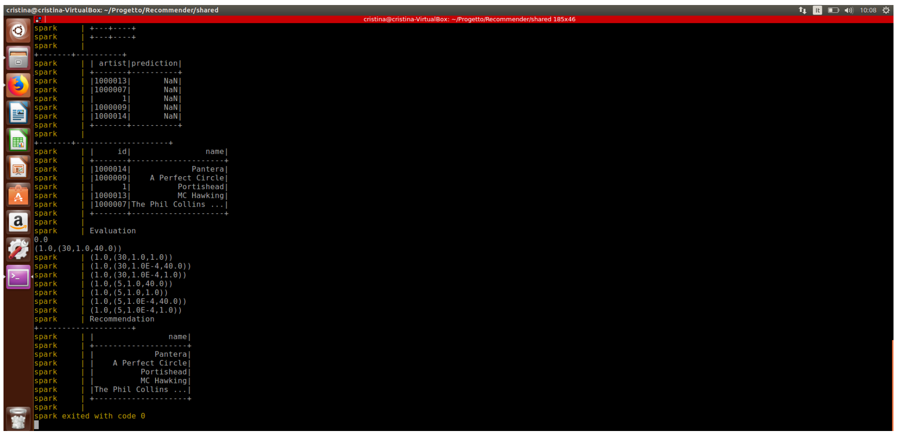
At this point Hadoop detects the *end.txt* file and stops too.

An example of execution is shown in Figure 1 and Figure 2.


cristina@cristina-VirtualBox: ~/Progetto/Recommender/shared
cristina@cristina-VirtualBox: ~/Progetto/Recommender/shared 185x46
10:08

```
ry for your platform... using builtin-java classes where applicable
hadoop     | starting yarn daemons
hadoop     | starting resourcemanager, logging to /usr/local/hadoop/logs/yarn--resourcemanage
r-432e34540ad0.out
hadoop     | localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nod
emanager-432e34540ad0.out
hadoop     | Adding dataset to Hadoop...
hadoop     | 18/01/11 04:03:01 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
hadoop     | Safe mode is OFF
hadoop     | 18/01/11 04:03:05 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
hadoop     | Creating directory...
hadoop     | 18/01/11 04:03:06 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
hadoop     | 18/01/11 04:03:08 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
hadoop     | Adding user_artist_data.txt...
hadoop     | 18/01/11 04:03:10 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
hadoop     | 18/01/11 04:03:12 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
hadoop     | Adding artist_data.txt...
hadoop     | 18/01/11 04:03:14 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
hadoop     | 18/01/11 04:03:17 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
hadoop     | Adding artist_alias.txt...
hadoop     | 18/01/11 04:03:19 WARN util.NativeCodeLoader: Unabl
e to load native-hadoop library for your platform... using buil
tin-java classes where applicable
spark      | Ziggy is running the Recommender...
spark      | Preparation
spark      | 1000002 1 55
spark      | 1000002 1000006 33
spark      | 1000002 1000007 8
spark      | 1000002 1000009 144
spark      | 1000002 1000010 314
spark      | +---------+---------+-----------+-----------+
spark      | |min(user)|max(user)|min(artist)|max(artist)|
spark      | +---------+---------+-----------+-----------+
spark      | |  1000002|  1000002|          1|    1000025|
spark      | +---------+---------+-----------+-----------+
spark      |
spark      | +-------+---------------+
spark      | |     id|           name|
```

Figure 1: Example of execution of the Recommender using Docker Compose


cristina@cristina-VirtualBox: ~/Progetto/Recommender/shared
cristina@cristina-VirtualBox: ~/Progetto/Recommender/shared 185x46
10:08

```
spark      | +---+----+
spark      | +---+----+
spark      |
+-------+----------+
spark      | | artist|prediction|
spark      | +-------+----------+
spark      | |1000013|       NaN|
spark      | |1000007|       NaN|
spark      | |      1|       NaN|
spark      | |1000009|       NaN|
spark      | |1000014|       NaN|
spark      | +-------+----------+
spark      |
+-------+--------------------+
spark      | |     id|                name|
spark      | +-------+--------------------+
spark      | |1000014|             Pantera|
spark      | |1000009|    A Perfect Circle|
spark      | |      1|          Portishead|
spark      | |1000013|          MC Hawking|
spark      | |1000007|The Phil Collins ...|
spark      | +-------+--------------------+
spark      |
spark      | Evaluation
0.0
(1.0,(30,1.0,40.0))
spark      | (1.0,(30,1.0,1.0))
spark      | (1.0,(30,1.0E-4,40.0))
spark      | (1.0,(30,1.0E-4,1.0))
spark      | (1.0,(5,1.0,40.0))
spark      | (1.0,(5,1.0,1.0))
spark      | (1.0,(5,1.0E-4,40.0))
spark      | (1.0,(5,1.0E-4,1.0))
spark      | Recommendation
+--------------------+
spark      | |                name|
spark      | +--------------------+
spark      | |             Pantera|
spark      | |    A Perfect Circle|
spark      | |          Portishead|
spark      | |          MC Hawking|
spark      | |The Phil Collins ...|
spark      | +--------------------+
spark      |
spark exited with code 0
```

Figure 2: Example of execution of the Recommender using Docker Compose