TDT4305 Big Data: Project - Phase 1

By Harald Aarskog and Frida Strand Kristoffersen

In this report, we will explain and describe briefly the Spark functions we have chosen to use during phase 1 of the project.

Creating a RDD (exercise 1-9)

```
import pyspark as py
from pyspark import SparkConf
from pyspark.context import SparkContext

sc = SparkContext.getOrCreate(SparkConf().setMaster("local[*]"))

artists = sc.textFile("/Users/fridastrandkristoffersen/Downloads/datasets/artists.csv")

artistlist = artists.map(lambda line: line.split(","))
```

In order to create a RDD we imported SparkContext from pyspark.context and used the function getOrCreate to instantiate a SparkContext and register it as a singleton object sparkConf allows you to configure some of the common properties (e.g. master URL and application name). setMaster.("local[*]") enables us to run Spark locally with as many worker threads as logical cores on our machine. Then we used the singleton object to register artists.csv as a RDD.

Creating a data frame (exercise 10)

```
import pyspark as py
from pyspark import SparkConf
from pyspark.context import SparkContext
from pyspark.sql import SQLContext
from pyspark.sql import Row
import csv

sc = SparkContext.getOrCreate(SparkConf().setMaster("local[*]"))

sqlc=SQLContext(sc)

#Loading the album dataset and converting it to a dataframe
album_df = sqlc.read.csv("/Users/haraldaarskog/Google\ Drive/Big\ data-arkitektur/Prosjekt/datasets/albums.csv")
album_df2 = album_df.toDF("id","artist_id","album_title","genre","year_of_pub",\
"num_of_tracks","num_of_sales","rolling_stone_critic","mtv_critic","music_maniac_critic")
#Registering the dataframe as a table
sqlc.registerDataFrameAsTable(album_df2, "albumTable")
```

We used the same function to create a Spark context as we did to create a RDD ("getOrCreate" and SparkConf()). In order to create a SQL context, we had to import the package pyspark.sql with SQLContext. Using the latter function with the Spark context object as an input, we can easily use the Spark context functions to read the csv files and then transforming it into a data frame. To make the data frame more convenient to work with, we registered the given data frame as a temporary table in the catalog using the SQL context function.

Spark RDD functions (exercise 1-9)

We classify the RDD functions in two: Transformations and actions. The transformation functions construct a new RDD from the previous one and the action functions compute a

result and store it in a file or return it to the driver program. We have used both transformation and action functions during phase 1 of the project.

Transformation

parallelize

- This Spark function was used to construct a new RDD from an existing collection of objects in the driver program. Thus we could easily write our results to a tsv-file using the RDD function saveAsTextFile() afterward.
- E.g. we had to use parallelize in exercise 6 after using the action “take(10)” where we took out the last ten elements in the RDD.

map

- Map applies a function on each element in a RDD and was one of the most used spark functions during this project.
- E.g. in exercise 1 we used this to create a RDD containing only the genre-column in the RDD called “albums” and in general it was really simple to generate a new RDD where only a few columns were included from the original. In exercise 6 we filtered out the album ID and registered it as the key in the new RDD and found the average among three critics, which the map function calculated for every row in the “albums” dataset.

reduceByKey

- This function performs the same operation on all values with the same key (a transformation, so it returns a RDD).
- E.g. in exercise 3 we used the map function to create a key-value pair of each country together with a 1. Then the reduceByKey could easily reduce the RDD into a list of each distinct country and summing up all the 1’s belonging to each country/key. We used exactly the same method for counting the total sales per genre in exercise 5.

sortBy()

- We used sortBy when we wanted to sort the elements within the RDD. It is possible to rank the ranges if you want a primary sorting column and if some of the elements are equal you can specify the secondary sorting column, as we did in exercise 4 where the primary column is the number of albums and the secondary sorting column is artist ID. One can use the True/False in order to specify whether we want to sort by ascending or descending order.
- E.g. in exercise 2 we used sortBy to sort the birth dates in ascending order.

join

- The join function joins two different RDDs based on the common keys in each RDD.
- E.g. we used join in exercise 7 in order to join the artist_country RDD with the album_top10 where both of the lists had the artist ID as their key.

distinct

- Distinct removes the duplicates in the RDD.
- E.g. we used in exercise 1 where genres.distinct().count() returns the number of genres in the albums RDD.

filter

- Filter filters out the elements which does not fulfill the requirements of the filter function. Thus it returns only a RDD consisting of only elements that pass the condition passed to the function.
- E.g. we used this function in exercise 8 where we filter the albums RDD on the MTV critics columns such that we end up with a RDD only containing the albums with an MTV critic equal to 5.

coalesce(XX)

- We used this function in order to avoid file partitioning. Thus it reduces the number of partitions in a RDD and if XX equals 1, we are guaranteed that the RDD is written out to disk as one text file.
- This was used in exercise [3,9]. I.e. all exercises where we wrote the RDD out to disk.

Actions

count
- Count counts the number of elements in the RDD
- E.g. we used count in exercise 1 in order to count the distinct genres in the albums RDD.

collect
- We used collect only as a debug tool while we were programming as it helped us figure at what the RDD's looked like.

saveAsTextFile
- Writes the RDD elements to a text file.
- This function is used in exercise [3,9].

take(xx)
- Takes the xx last elements from a RDD and return a result to store in a file or to return to the driver program.
- We used this function in exercise 2 in order to take out the oldest birth year after using sortBy to make sure that the oldest birth year was in the last element in the list.
- The function was also used in exercise 6 to take out the top 10 albums with the highest average critic.

Spark DataFrame functions (exercise 10)

After we converted the two datasets into data frames and registering them as tables, we chose to use SQL-functions on the data frame tables. Thus we have chosen to explain these briefly as well even though it is a part of the prerequisites for this course.

Select
- Selects the columns that we want to return
- Used in all sub exercises

From
- Specifies which table(s) we want to use during the query
- Used in all sub exercises

Count
- This function counts the number of elements in a given table.
- We used this function in e.g. exercise 10a together with distinct in order to count the number of distinct artists (based on the artist id)

Distinct
- Distinct filters out the elements that already exist in the table such that we are left only distinct elements.

MIN
- Returns the element with the lowest value
- E.g. we used this function in exercise 10e to find the first year an album in the albums dataset was published

MAX
- Returns the element with the highest value
- E.g. we used this function in exercise 10h in order to find the year the youngest artist in artists dataset was born.

show()

- The action show was used in order to show the top 20 rows of the data frame in a tabular form. Since we never had 20 elements in our output table, we could use this action to print out the whole output table for each sub exercise.