Spark Project

April 24, 2018

1 Spark Assignment 2018

1.1 Exercise 1

For each exercise, the first cell creates the function and the second cell executes the query and prints the result.

1.1.1 Exercise 1.1.

```
In [1]: from pyspark.sql import SQLContext
        import pandas as pd
        from pyspark.sql.functions import mean, min, max, col, avg, monotonically_increasing_i
        import pyspark.sql.functions as F
        from operator import add
        import time

        file = "pagecounts-20160101-000000"
        sqlContext = SQLContext(sc)
        rdd = sc.textFile(file)

        def f1(rdd):
            input_data = rdd.map(lambda line: line.split(" "))
            df_normal = input_data.toDF()

            # appearance
            df= df_normal.limit(15).toPandas()
            for index, row in df.iterrows():
                print(row[0], row[1], row[2], row[3])
```

```
In [4]: f1(rdd)
```

```
aa %CE%92%CE%84_%CE%95%CF%80%CE%B9%CF%83%CF%84%CE%BF%CE%BB%CE%AE_%CE%99%CF%89%CE%AC%CE%BD%CE%B
aa %CE%98%CE%B5%CF%8C%CE%B4%CF%89%CF%81%CE%BF%CF%82_%CE%91%CE%84_%CE%9B%CE%AC%CF%83%CE%BA%CE%B
aa %CE%9C%CF%89%CE%AC%CE%BC%CE%B5%CE%B8_%CE%95%CE%84/el/%CE%9C%CE%B5%CF%87%CE%BC%CE%AD%CF%84_%
aa %CE%A0%CE%B9%CE%B5%CF%81_%CE%9B%27_%CE%91%CE%BD%CF%86%CE%AC%CE%BD/el/%CE%A0%CE%B9%CE%B5%CF%
aa %CE%A3%CE%A4%CE%84_%CE%A3%CF%84%CE%B1%CF%85%CF%81%CE%BF%CF%86%CE%BF%CF%81%CE%AF%CE%B1/el/%C
aa %D0%A1%D0%BE%D0%BB%D0%B8_484_%D0%BF.%D0%BC 1 4750
aa 271_a.C 1 4675
aa Battaglia_di_Qade%C5%A1/it/Battaglia_dell%27Oronte 1 4765
aa Category:User_th 1 4770
aa Chiron_Elias_Krase 1 4694
aa County_Laois/en/Queen%27s_County,_Ireland 1 4752
aa Dassault_rafaele 2 9372
aa Dyskusja_wikiprojektu:Formu%C5%82a_1/%22/pl/Polacy_w_Formule_1%22 1 4824
aa E.Desv 1 4662
aa Enclos-apier/fr/Enclos-Apiers_en_C%C3%B4te_d%27Azur 1 4772
```

1.1.2 Exercise 1.2.

```
In [5]: def f2(rdd):
            input_data = rdd.map(lambda line: line.split(" "))
            df_normal = input_data.toDF(['code','title','hits', 'size'])
            print(df_normal.count())
```

```
In [6]: f2(rdd)
```

```
5046226
```

1.1.3 Exercise 1.3.

```
In [2]: def f3(rdd):
            input_data = rdd.map(lambda line: line.split(" "))
            df_normal = input_data.toDF(['code','title','hits', 'size'])

            # convert to double
            df_normal = df_normal.withColumn('size', col('size').cast('double'))
            df_normal = df_normal.withColumn('hits', col('hits').cast('double'))

            # get stats
            x = df_normal.select([mean('size'), min('size'), max('size')]).collect()
            result = "Mean = " + str(x[0][0]) + "; Min = " + str(x[0][1])+ "; Max = " + str(x[
            return result
```

```
In [8]: print(f3(rdd))
```

```
Mean = 101423.92964801814; Min = 0.0; Max = 141180155987.0
```

1.1.4 Exercise 1.4.

```
In [9]: def f4(rdd):
            input_data = rdd.map(lambda line: line.split(" "))
            df_normal = input_data.toDF(['code','title','hits', 'size'])

            # convert to double from string
            df_normal = df_normal.withColumn('size', col('size').cast('double'))
            df_normal = df_normal.withColumn('hits', col('hits').cast('double'))

            #get max
            max_value = df_normal.select(max('size')).collect()[0][0]
            l=[max_value]
            df = df_normal.where(df_normal.size.isin(l))

            df= df.toPandas()
            for index, row in df.iterrows():
                print(row[0], row[1], row[2], row[3])
```

```
In [10]: f4(rdd)
```

```
en.mw en 5466346.0 141180155987.0
```

Only one record with maximum page size value.

1.1.5 Exercise 1.5.

```
In [3]: def f5(rdd):
            input_data = rdd.map(lambda line: line.split(" "))
            df_normal = input_data.toDF(['code','title','hits', 'size'])

            # convert to double from string
            df_normal = df_normal.withColumn('size', col('size').cast('double'))
            df_normal = df_normal.withColumn('hits', col('hits').cast('double'))

            # get max
            max_value = df_normal.select(max('size')).collect()[0][0]
            l=[max_value]
            df = df_normal.where(df_normal.size.isin(l))

            # pick most hits if several records
            max_hits = df.select(max('hits')).collect()[0][0]
            l=[max_hits]
            df = df_normal.where(df.hits.isin(l))

            return df
```

```
In [12]: df = f5(rdd).toPandas()
         for index, row in df.iterrows():
             print(row[0], row[1], row[2], row[3])
```

```
en.mw en 5466346.0 141180155987.0
```

1.1.6 Exercise 1.6.

```
In [4]: def f6(rdd):
            input_data = rdd.map(lambda line: line.split(" "))
            df_normal = input_data.toDF(['code','title','hits', 'size'])

            # create title length column
            new = df_normal.withColumn("length", F.length('title'))

            # get longest page title
            max_value = new.select(max('length')).collect()[0][0]
            l=[max_value]
            df = new.where(new.length.isin(l))

            return df
```

Result of query is commented out because query is very long. Please remove "#" to execute query.

```
In [15]: #df = f6(rdd).toPandas()
         #for index, row in df.iterrows():
             #print(row[1])
             #print()
```

1.1.7 Exercise 1.7.

```
In [5]: def f7(rdd):
            input_data = rdd.map(lambda line: line.split(" "))
            df_normal = input_data.toDF(['code','title','hits', 'size'])

            # convert to double from string
            df_normal = df_normal.withColumn('size', col('size').cast('double'))
            df_normal = df_normal.withColumn('hits', col('hits').cast('double'))

            # get mean
            mean_value = df_normal.select(mean('size')).collect()[0][0]

            # filter
            df = df_normal.filter(df_normal.size > mean_value)

            df = df.rdd
            return df
```

```
In [16]: f7(rdd).take(5)
```

```
Out[16]: [Row(code='aa', title='Main_Page', hits=5.0, size=266946.0),
          Row(code='ab', title='%D0%90%D0%B2%D0%B8%D0%BA%D0%B8%D0%BF%D0%B5%D0%B4%D0%B8%D0%B0:%
          Row(code='ab', title='%D0%90%D2%B3%D3%99%D1%8B%D0%BD%D2%AD%D2%9B%D0%B0%D1%80%D1%80%D
          Row(code='ab', title='%D0%91%D1%80%D0%B8%D1%82%D0%B0%D0%BD%D0%B8%D0%B0_%D0%94%D1%83_
          Row(code='ab', title='%D0%92%D0%BB%D0%B0%D0%B4%D0%B8%D0%BC%D0%B8%D1%80_%D0%9F%D1%83%
```

1.1.8 Exercise 1.8.

```
In [17]: def f8(rdd):
             # Compute the total number of pageviews for each project
             input_data = rdd.map(lambda line: line.split(" "))
             df_normal = input_data.toDF(['code','title','hits', 'size'])

             # convert to double from string
             df_normal = df_normal.withColumn('size', col('size').cast('double'))
             df_normal = df_normal.withColumn('hits', col('hits').cast('double'))

             # group by project code
             sum = df_normal.groupBy("code").sum("hits").toPandas()

             for index, row in sum.iterrows():
                 print("Total number of page views for \"" + str(row[0]) + "\" project = " + s
```

Result of query is commented out because query is very long. Please remove "#" to execute query.

```
In [13]: #f8(rdd)
```

1.1.9 Exercise 1.9.

```
In [19]: def f9(rdd):
             # Report the 10 most popular pageviews of all projects , sorted by the total numb
             input_data = rdd.map(lambda line: line.split(" "))
             df_normal = input_data.toDF(['code','title','hits', 'size'])

             # convert to double from string
             df_normal = df_normal.withColumn('size', col('size').cast('double'))
             df_normal = df_normal.withColumn('hits', col('hits').cast('double'))

             query = df_normal.orderBy(['hits'], ascending=False)

             # clean visualization
             df= query.limit(10).toPandas()
             for index, row in df.iterrows():
                 print(row[0], row[1], row[2], row[3])
```

```
In [20]: f9(rdd)
```

```
en.mw en 5466346.0 141180155987.0
es.mw es 695531.0 12261337515.0
ja.mw ja 611443.0 15021588551.0
de.mw de 572119.0 9523069696.0
fr.mw fr 536978.0 11752030020.0
ru.mw ru 466742.0 11847816616.0
it.mw it 400297.0 8176042087.0
```

```
en Main_Page 257915.0 4289970372.0
pt.mw pt 196160.0 4029404403.0
pl.mw pl 176059.0 2782453516.0
```

1.1.10 Exercise 1.10.

```
In [21]: def f10(rdd):
             input_data = rdd.map(lambda line: line.split(" "))
             df_normal = input_data.toDF(['code','title','hits', 'size'])

             # Determine the number of page titles that start with the article The.
             # How many of those page titles are not part of the English project?
             query = df_normal.withColumn('The', df_normal['title'].substr(0, 3)=='The')
             query = query.filter(query.The == True)
             print("Number of titles that start with The = "+str(query.count()))
             query = query.filter(query.code!='en')
             print("Number of titles that start with The and are not in \"en\" = "+str(query.co
```

```
In [22]: f10(rdd)
```

```
Number of titles that start with The = 48684
Number of titles that start with The and are not in "en" = 11553
```

1.1.11 Exercise 1.11.

```
In [23]: def f11(rdd):
             input_data = rdd.map(lambda line: line.split(" "))
             df_normal = input_data.toDF(['code','title','hits', 'size'])

             # convert to double from string
             df_normal = df_normal.withColumn('size', col('size').cast('double'))
             df_normal = df_normal.withColumn('hits', col('hits').cast('double'))

             # Determine the percentage of pages that have only received a single page view in
             query = df_normal.filter(df_normal.hits == 1).count()
             print(str(round(100*query/df_normal.count(),2))+"%")
```

```
In [24]: f11(rdd)
```

```
79.38%
```

1.1.12 Exercise 1.12.

```
In [29]: def f12(rdd):
             input_data = rdd.map(lambda line: line.split(" "))
             df_normal = input_data.toDF(['code','title','hits', 'size'])
```

```
            # Determine the number of unique terms appearing in the page titles. Note that in
            # titles, terms are delimited by "_" instead of a whitespace.
            df = df_normal.select(F.split(F.col("title"), "_")).rdd.map(lambda r : r[0])
            print(df.flatMap(set).distinct().count())
```

```
In [30]: f12(rdd)
```

```
3491232
```

1.1.13 Exercise 1.13.

```
In [6]: def f13(rdd):
            # Determine the most frequently occurring page title term in this dataset.
            input_data = rdd.map(lambda line: line.split(" "))
            df_normal = input_data.toDF(['code','title','hits', 'size'])

            df = df_normal.select(F.split(F.col("title"), "_")).rdd.map(lambda r : r[0])
            df = df.flatMap(lambda doc: [(x, 1) for x in doc]).reduceByKey(add)
            df = df.toDF(['word', 'count'])
            x = df.select([max('count')]).collect()[0][0]
            return x
```

```
In [28]: print(f13(rdd))
```

```
215677
```

1.2 Exercise 2

```
In [7]: # set up
        sc.stop()
        conf = SparkConf().setAppName('appName').setMaster('spark://Jeans-MBP:7077')
        sc = SparkContext(conf=conf)

        # define rdd
        sqlContext = SQLContext(sc)
        rdd = sc.textFile(file)
```

1.2.1 Small cluster configuration:

SPARK_WORKER_CORES=2
SPARK_WORKER_INSTANCES=2
SPARK_WORKER_MEMORY=1g

```
In [11]: # Query 3
         start_time = time.time()
         f3(rdd)
         print("Query 3 takes %s seconds" % (time.time() - start_time))

         # Query 5
         start_time = time.time()
         f5(rdd)
         print("Query 5 takes %s seconds" % (time.time() - start_time))

         # Query 6
         start_time = time.time()
         f6(rdd)
         print("Query 6 takes %s seconds" % (time.time() - start_time))

         # Query 7
         start_time = time.time()
         f7(rdd)
         print("Query 7 takes %s seconds" % (time.time() - start_time))

         # Query 13
         start_time = time.time()
         f13(rdd)
         print("Query 13 takes %s seconds" % (time.time() - start_time))
```

```
Query 3 takes 19.41314172744751 seconds
Query 5 takes 44.53948783874512 seconds
Query 6 takes 18.51176691055298 seconds
Query 7 takes 16.548261880874634 seconds
Query 13 takes 69.00603985786438 seconds
```

1.2.2 Big cluster configuration:

SPARK_WORKER_CORES=8
SPARK_WORKER_INSTANCES=8
SPARK_WORKER_MEMORY=8g

```
In [12]: # Query 3
         start_time = time.time()
         f3(rdd)
         print("Query 3 takes %s seconds" % (time.time() - start_time))

         # Query 5
         start_time = time.time()
         f5(rdd)
         print("Query 5 takes %s seconds" % (time.time() - start_time))

         # Query 6
         start_time = time.time()
         f6(rdd)
```

```
    print("Query 6 takes %s seconds" % (time.time() - start_time))

    # Query 7
    start_time = time.time()
    f7(rdd)
    print("Query 7 takes %s seconds" % (time.time() - start_time))

    # Query 13
    start_time = time.time()
    f13(rdd)
    print("Query 13 takes %s seconds" % (time.time() - start_time))
```

```
Query 3 takes 18.175821781158447 seconds
Query 5 takes 36.74556493759155 seconds
Query 6 takes 17.101213932037354 seconds
Query 7 takes 15.644676923751831 seconds
Query 13 takes 60.782407999038696 seconds
```

"SPARK_WORKER_CORES" is the number of cores per worker. "SPARK_WORKER_INSTANCES" is the number of individual workers in the cluster. "SPARK_WORKER_MEMORY" is the amount of memory allocated to each cluster.

The big cluster takes significantly more time to compute the queries 3, 5 and 13. This is because as one adds more and more nodes to the cluster, communication cost increases and therefore computing time increases. For queries 6 and 7 the computing time is about the same probably because the queries require less computation power and therefore the extra computation power of the big cluster is not exploited.