Zafar Mahmood
Solution Exercise Sheet 7
30/May

Question 1 : rightOuterJoin and fullOuterJoin

As the join always happen on key value.

so we can't call join operation so have to convert into key value pairs so after converting it give the following solution for rightOuterJoin

```
Array[(String, (Option[Int], Int))] = Array((python,(Some(3),3)), (class,(Some(6),6)), (create,
(Some(5),5)), (spark,(Some(1),1)), (rdd,(Some(2),2)), (context,(Some(4),4)))
```

Now fullOuterJoin

```
 Array[(String, (Option[Int], Option[Int]))] = Array((python,(Some(3),Some(3))), (class,
(Some(6),Some(6))), (create,(Some(5),Some(5))), (spark,(Some(1),Some(1))), (rdd,(Some(2),Some(2))),
(context,(Some(4),Some(4))))
```

Question 2 : Using Map Reduce count's

```
a = sc.parallize (List( "spark", "rdd", "python", "context", "create", "class"))
b = sc.parallize (List ( "operation", "apache", "scala", "lambda","parallel","partition"))
```

1) first step

```
c = a.union(b)
```

2) Counts the characters occurring

```
var counts = c.flatMap(ctr => ctr.split("")).map(sg => ( sg, 1)).reduceByKey(_+_)
```

As the map function will send character with 1 count and reduce function will count each characters

```
Array[(String, Int)] = Array((p,6), (x,1), (h,2), (a,13), (i,3), (y,1), (b,1), (r,6), (s,4), (k,1),
(c,5), (d,3), (t,7), (l,6), (e,6), (m,1), (n,4), (o,5))
```

Here as it show's s to be counted 4 times

Question 3 : Aggregate function

```
rdd_1 = ["spark", "rdd", "python", "context", "create", "class"]
>>> rdd_2 = ["operation", "apache", "scala", "lambda","parallel","partition"]
>>> rdd_1 = sc.parallelize(rdd_1)
>>> rdd_2 = sc.parallelize(rdd_2)
>>> union_rdd = rdd_1.union(rdd_2)
>>> result_addgregate = union_rdd.flatMap(lambda word: word).map(lambda letter: (letter,1)).aggregateByKey(0,
(lambda k,v:v+k),(lambda v,k:v+k))
>>> result_addgregate
PythonRDD[7] at RDD at PythonRDD.scala:48
>>> result_addgregate = union_rdd.flatMap(lambda word: word).map(lambda letter: (letter,1)).aggregateByKey(0,
(lambda k,v:v+k),(lambda v,k:v+k)).count()
>>> result_addgregate
18
>>> result_addgregate = union_rdd.flatMap(lambda word: word).map(lambda letter: (letter,1)).aggregateByKey(0,
(lambda k,v:v+k),(lambda v,k:v+k))
```

Exercise 1B : Spark Question

Part A : Your task is to first separate out tagging sessions for each user

```
from pyspark import SparkContext
import numpy as np

sc = SparkContext("local","Simple App")

logData = sc.textFile("tags.dat")

extracted_userData = logData.map(lambda x:x.split('::')).map(lambda y:
(int(y[0]),str(y[2]),int(y[3]))).collect()
```

```
>>> red_new[1:20]
[(20, 'politics', 1188263867), (20, 'satire', 1188263867), (20, 'chick flick 212', 1188263835), (20,
'hanks', 1188263835), (20, 'ryan', 1188263835), (20, 'action', 1188263755), (20, 'bond',
1188263756), (20, 'spoof', 1188263880), (20, 'star wars', 1188263880), (20, 'bloody', 1188263801),
(20, 'kung fu', 1188263801), (20, 'Tarantino', 1188263801), (21, 'R', 1205081506), (21, 'NC-17',
1205081488), (25, 'Kevin Spacey', 1166101426), (25, 'Johnny Depp', 1162147221), (31, 'buddy comedy',
1188263759), (31, 'strangely compelling', 1188263674), (31, 'catastrophe', 1188263741)]
```

Now as the mapping function will gather all the data , here I showed userID and Tag and seconds. Now the second part of question

Tagging Sessions

as each session is denoted by 30 mins window, every tagging that's happening within this session is considered as single session tagging frequency.

Algorithm

30 mins = 1800 second

Now each time stamp is in second (from UT 12:00)

Calculation of session

```
if timestamp[i] < presentTimeStamp+1800 :
      sameSession
      frequency ++
else :
      differentSession
      frequency = 1
```

Part B : standard deviation between their tagging frequency per session and mean

Now we have each user time-stamp , with their tagging frequencies in a list all we have to do is to calculate mean, frequencies and standard Deviation upon a certain threshold

Sample Output :

```
UserID 13688, TagginSession 5, STD 7.6, mean 4.8,
UserID 13701, TagginSession 2, STD 2.5, mean 7.5,
UserID 13880, TagginSession 3, STD 2.16024689947, mean 6.0,
UserID 14149, TagginSession 3, STD 8.80656320908, mean 9.66666666667,
UserID 14208, TagginSession 17, STD 5.40408662487, mean 6.17647058824,
UserID 14343, TagginSession 3, STD 2.16024689947, mean 3.0,
UserID 14421, TagginSession 8, STD 11.3516243331, mean 13.125,
UserID 14578, TagginSession 10, STD 20.4978047605, mean 15.8,
UserID 14598, TagginSession 19, STD 2.87216076835, mean 2.47368421053,
```

Code :

```
from pyspark import SparkContext
import numpy as np

user_time_stamp = []
avg_user_stamp = []
user = []

sc = SparkContext("local","Simple App")

logData = sc.textFile("tags.dat")

extracted_userData = logData.map(lambda x:x.split('::')).map(lambda y:
(int(y[0]),int(y[3]))).collect()

print(len(extracted_userData))

mat = np.matrix(extracted_userData)

v = len(np.array(mat[:,0]))
print("length of v ", v)
unique_users = np.zeros(v)
time_stamps = np.zeros(v)
array_counter = 0
all_time_stamps = []
presentID = 0
counter = 0

for i in range(0,v):
    _id = mat[i,0]
    #print(_id)

    if(counter == 0):
        counter += 1
        print("this is counter")
        presentID = _id
        array_counter += 1
        user.append(presentID)
        user_time_stamp.append(mat[i,1])

    else :

        if (presentID != _id ):
            user.append(presentID)
            all_time_stamps.append(list(np.sort(user_time_stamp)))
            user_time_stamp = []
            array_counter += 1
            presentID = _id

        user_time_stamp.append(mat[i,1])



print(array_counter)


#user.append(presentID)
all_time_stamps.append(list(np.sort(user_time_stamp)))
user_time_stamp = []

print("shape of stamps ",np.shape(all_time_stamps))
print("few data",all_time_stamps[0:10])
```

```
print(type(all_time_stamps))

## Now next questions

user_tagging_frequecies = []
user_tagging_time_stamps = []

taggin_session = 1800

temp_stamp = 0,
temp_frequency_session = 0

_fileWrite = open("taging.txt","w")

for i in range(0,len(user)):
    print("lenght:", len(all_time_stamps[i]))

    for j in range(0,len(all_time_stamps[i])):

        if (len(all_time_stamps[i]) == 1):
            user_tagging_frequecies.append(1)
        else :

            if (j == 0):
                temp_stamp = all_time_stamps[i][j]
                temp_frequency_session = 1
            else :
                if(all_time_stamps[i][j] >= (temp_stamp + 1800)):

                    user_tagging_frequecies.append(temp_frequency_session)
                    temp_stamp = all_time_stamps[i][j]
                    temp_frequency_session = 1
                else :
                   temp_frequency_session += 1


    if (temp_frequency_session != 0):
        user_tagging_frequecies.append(temp_frequency_session)

    #user_tagging_frequecies.append(temp_frequency_session)

    print("user ID ",user[i], " tagging_sessions:", len(user_tagging_frequecies)," 
std:",np.std(user_tagging_frequecies)," mean:",np.mean(user_tagging_frequecies))
    print(user_tagging_frequecies)
    if (np.std(user_tagging_frequecies)>2):
        print("user ID ",user[i], " tagging_sessions:", len(user_tagging_frequecies)," 
std:",np.std(user_tagging_frequecies)," mean:",np.mean(user_tagging_frequecies), "\n")
        _fileWrite.write("UserID %s, TagginSession %s, STD %s, mean %s,\n" % (user[i], 
len(user_tagging_frequecies), np.std(user_tagging_frequecies), np.mean(user_tagging_frequecies)))
    user_tagging_frequecies = []
    temp_frequency_session = 0
    if (i == 1000):
        break
```

Output File is also attached “Taging.txt”

Exercise 2 : Running Apache Spark on HDFS using Yarn

a) Find the user who has assign highest average rating among all the users who rated less than 60 times

Solution

Algorithm

The algorithm is the same old one ,

is to first use the map function and then use present & Old user ID to work with rating,

Now after each change of userID check check if he is rated less than 60 times. If so , then calculate the average and find the highest average user.

b) **An interesting feature one wants to know is the time of the day when a particular user is likely to rate a movie. Please find time of the day when a given user will rate movies.**

Algorithm

For this algorithm ,

As we have the user and tagging of each user’s each time

Tagging stamp = UTC from 12:00 ~ 1 Jan 1970

First I will calculate the mean tagging time, by merging all of its timestamps and taking a mean

now using time function to calculate its HH:MM:SS

Now here, hours will determine weather its time.

Firstly I have was able to configure YARN with HADOOP

and tested pi.py example and it worked perfectly fine in 34 second approx

Configuration

yarn.xml

```
<property>
   <name>yarn.nodemanager.aux-services</name>
   <value>mapreduce_shuffle</value>
 </property>
 <property>
   <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
   <value>org.apache.hadoop.mapred.ShuffleHandler</value>
 </property>

<property>
  <name>yarn.nodemanager.delete.debug-delay-sec</name>
  <value>1200</value>
</property>
```

spark-env.sh

HADOOP_CONF_DIR=/home/zfar/hadoop-2.7.2/etc/hadoop

to make this variable available for environment

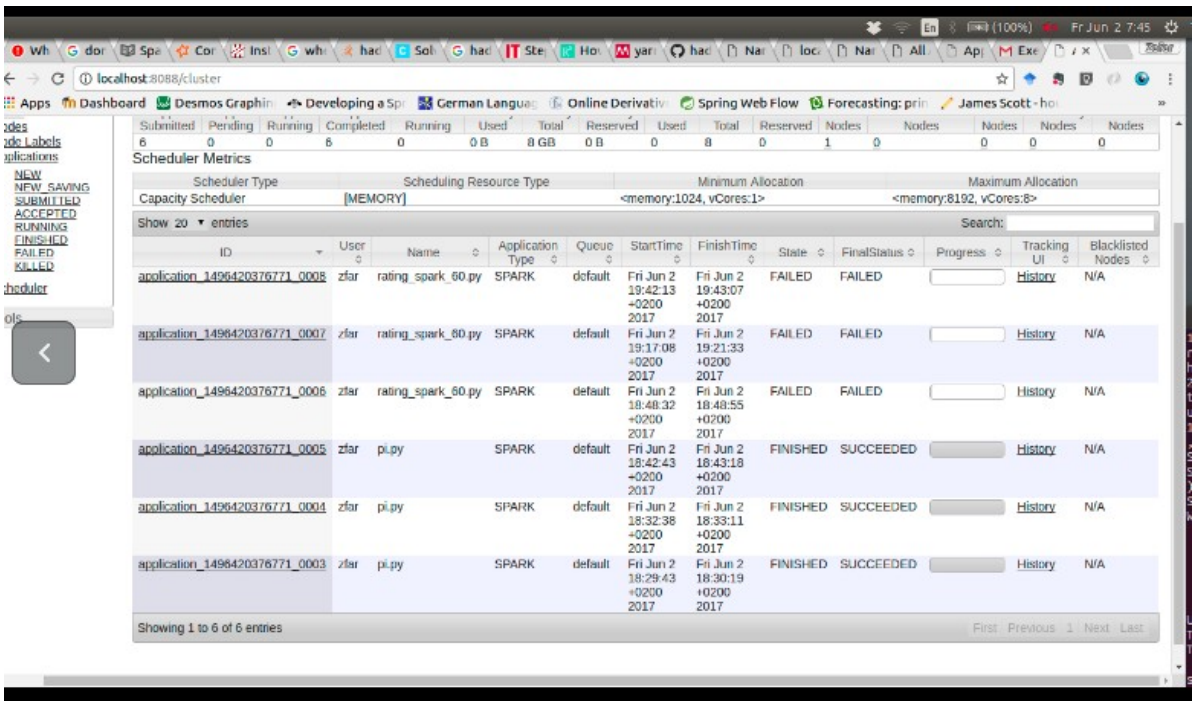
Now run this command to for testing in cluster (YARN) => Pi.py

$ spark-submit --master yarn --deploy-mode cluster --executor-cores 2 pi.py 10 (worked perfectly fine)

Now After this when I run my present python files, continuously received errors of AM container failed two times , **memory error**

And even after this when I run back the pi.py file it was taking too much time to process in YARN.

So the



localhost:8088/cluster

Apps | Dashboard | Desmos Graphin | Developing a Spr | German Languag | Online Derivativ | Spring Web Flow | Forecasting: prin | James Scott - hou

Connection Established
You are now connected to the Wi-Fi network 'eduroam'.

hadoop

All Applications

Logged in as: dr.who

Cluster
- About
- Nodes
- Node Labels
- Applications
  - NEW
  - NEW_SAVING
  - SUBMITTED
  - ACCEPTED
  - RUNNING
  - FINISHED
  - FAILED
  - KILLED
- Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
3	0	0	3	0	0 B	8 GB	0 B	0	8	0	1	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1496426409455_0003	zfar	rating_spark_60.py	SPARK	default	Fri Jun 2 20:19:21 +0200 2017	Fri Jun 2 20:19:47 +0200 2017	FAILED	FAILED		History	N/A
application_1496426409455_0002	zfar	pi.py	SPARK	default	Fri Jun 2 20:12:25 +0200 2017	Fri Jun 2 20:14:36 +0200 2017	FINISHED	SUCCEEDED		History	N/A
application_1496426409455_0001	zfar	rating_spark_60.py	SPARK	default	Fri Jun 2 20:01:44 +0200 2017	Fri Jun 2 20:04:04 +0200 2017	FAILED	FAILED		History	N/A

Showing 1 to 3 of 3 entries First Previous 1 Next Last