BIG DATA LAB 1

Rabnawaz Jansher & Saman Zahid

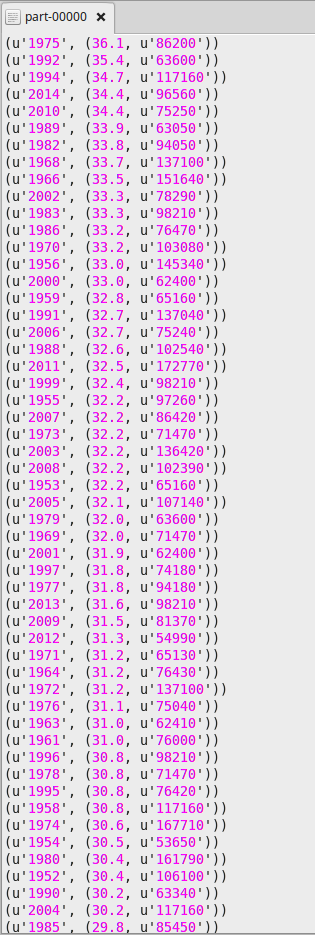
7 May 2018

# BDA1 -Spark -Exercises

# Question 1

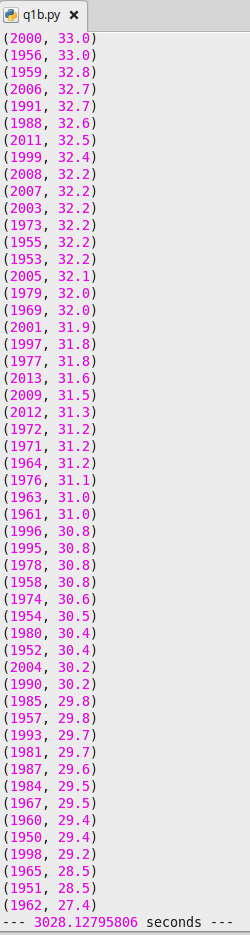
## Part A

from pyspark import SparkContext  
# function for calculating max temperature   
# also returns associated stations  
def max\_temperature(a,b):  
 if a[0]>=b[0]:  
 return a  
 else:  
 return b  
sc = SparkContext(appName = "lab1-q1")  
# reading data  
temperature\_file = sc.textFile("/user/x\_samza/data/temperature-readings.csv")  
# splitting columns  
lines = temperature\_file.map(lambda line: line.split(";"))  
# making key value pair keeping year as key and temp and station as values  
station\_year\_temperature = lines.map(lambda x: (x[1][0:4],(float(x[3]),x[0])))  
# filtering data between 1950 and 2014  
station\_year\_temperature = station\_year\_temperature.filter(lambda x: int(x[0]) >= 1950 and int(x[0]) <= 2014)  
# finding max\_temperature based on key - 'year'  
max\_temperatures = station\_year\_temperature.reduceByKey(lambda (x11,x21),(x12,x22): max\_temperature((x11,x21),(x12,x22)))  
# sorting result in descending order of max temperature  
max\_temperatureSorted = max\_temperatures.sortBy(ascending = False , keyfunc = lambda k : k[1][0])  
# saving result to hadoop  
max\_temperatureSorted.saveAsTextFile("max\_temperature\_station")



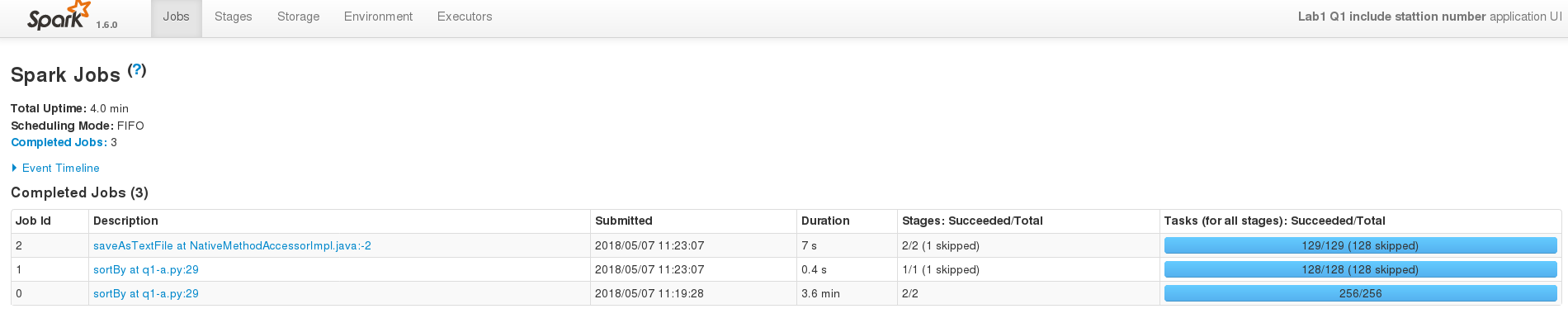
## Part B

#non parallel python program  
#import csv  
import csv  
import time  
start\_time = time.time()  
#dict   
data = {}  
# read data   
with open('/nfshome/hadoop\_examples/shared\_data/temperatures-big.csv') as csvDataFile:  
 #split data   
 csvReader = csv.reader(csvDataFile,delimiter=';')  
 for row in csvReader:  
 year = int(row[1][0:4])  
 #filter data and compare temperature   
 if int(year) >= 1950 and int(year) <= 2014:  
 temp = float(row[3])  
 if not data:  
 data[year] = temp  
 else:  
 if year in data.keys():  
 if data[year] < temp:  
 data[year] = temp  
 else:  
 data[year] = temp  
   
#sort data  
for row in sorted(data.items(),key=lambda x: (x[1],x[0]),reverse=True):  
 print row  
#print time  
print("--- %s seconds ---" % (time.time() - start\_time))



### comparison

Spark Program Execution time on temperature-big csv is 4 minutes 



Non Parallel python Program Execution

3028.12795806 second which is equal to 50 minutes

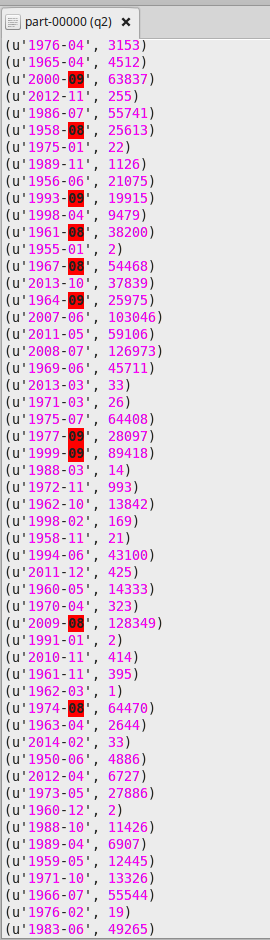
### Reason

It is because spark excute a program in parallel on distributed environment so its excution time is less than non parallel excution of program.

# Quesrion 2

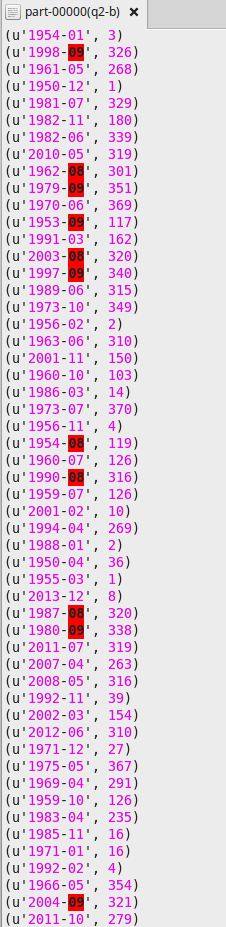
## Part A

#import spark libraries  
from pyspark import SparkContext  
from operator import add  
#spark context object  
sc = SparkContext(appName = "Lab1 Q2-count-records")  
#read temperature  
temperature\_file = sc.textFile("/user/x\_rabsh/data/temperature-readings.csv")  
lines = temperature\_file.map(lambda line: line.split(";"))  
#filter data year 1950 t0 2014 and temperature greater than 10  
temperature = lines.filter(lambda x: int(x[1][0:4])>=1950 and int(x[1][0:4])   
 <= 2014 and float(x[3]) > 10)  
year\_temperature = temperature.map(lambda x: (x[1][0:7], 1))  
count = year\_temperature.reduceByKey(add)  
#repartion data and save  
monthly\_temperatures\_count = count.repartition(1)  
monthly\_temperatures\_count.saveAsTextFile("lab1\_q2a")



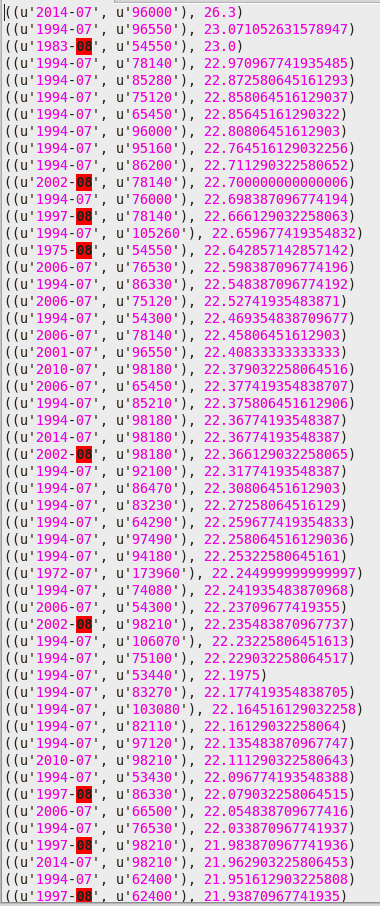
## Part B

from pyspark import SparkContext  
#count distinct elements  
def count\_distinct(a,b):  
 return (a[0], a[1]+b[1])  
   
sc = SparkContext(appName = "Lab1 Q2-distnict elements")  
#read a data  
temperature\_file = sc.textFile("/user/x\_rabsh/data/temperature-readings.csv")  
lines = temperature\_file.map(lambda line: line.split(";"))  
#filter temperature 1950 to 2014 and temperature greater than 10  
temperature = lines.filter(lambda x: int(x[1][0:4])>=1950 and int(x[1][0:4])  
 <= 2014 and float(x[3]) > 10)  
year\_temperature = temperature.map(lambda x: (x[1][0:7], (x[0],1) )).distinct()  
count = year\_temperature.reduceByKey(lambda v1, v2: count\_distinct(v1,v2))  
#count elements  
count = count.map(lambda x: (x[0], x[1][1]))   
#repartition data and save  
monthly\_temperatures\_count = count.repartition(1)  
monthly\_temperatures\_count.saveAsTextFile("lab1\_q2b")



# Quesrion 3

# importing spark context  
from pyspark import SparkContext  
# function for calculating max temperature  
def max\_temp(a,b):  
 if a >= b:  
 return a  
 else:  
 return b  
   
# function for calculating min temperature  
def min\_temp(a,b):  
 if a <= b:  
 return a  
 else:  
 return b  
# defining spar context   
sc = SparkContext(appName = "lab1-q3")  
# reading temperature data file  
temperature\_file = sc.textFile("/user/x\_samza/data/temperature-readings.csv")  
# splitting columns in data  
lines = temperature\_file.map(lambda line: line.split(";"))  
# generating key-value pair with year-month-date and station as key and temperature as value  
year\_temperature = lines.map(lambda x: ((x[1][0:10],x[0]),float(x[3])))  
# filtering data between 1950 and 2014  
year\_temperature = year\_temperature.filter(lambda x: int(x[0][0][0:4]) >= 1950 and int(x[0][0][0:4]) <= 2014)  
# Calculating max temperature  
max\_temperatures = year\_temperature.reduceByKey(max\_temp)  
# Calculating min temperature  
min\_temperature = year\_temperature.reduceByKey(min\_temp)  
# Joining max\_temperature RDD and min temperature RDD  
max\_min\_temperatures = max\_temperatures.join(min\_temperature)  
# Adding min and max temperature and associating count value to use later  
monthly\_temp = max\_min\_temperatures.map(lambda x: ((x[0][0][0:7],x[0][1]), (float(x[1][0])+float(x[1][1]),2) ))  
# Adding previous and current temperature and count on the basis of key  
monthly\_avg\_temp = monthly\_temp.reduceByKey(lambda (temp1, count1), (temp2, count2): (temp1+temp2, count1+count2))  
# mapping required columns and calculating avg by dividing the sum of daily avg by totat no. of days in month  
monthly\_avg\_temp = monthly\_avg\_temp.map(lambda ((date, station), (sumTemp, sumCount)): ((date, station), sumTemp/float(sumCount)))  
# repartitioning to form single RDD  
max\_temperature\_monthly= monthly\_avg\_temp.repartition(1)  
# Sorting result in descending order  
max\_temperature\_monthly = max\_temperature\_monthly.sortBy(ascending = False , keyfunc = (lambda k : k[1]))  
# Saving File  
max\_temperature\_monthly.saveAsTextFile("avg\_temperature\_station")



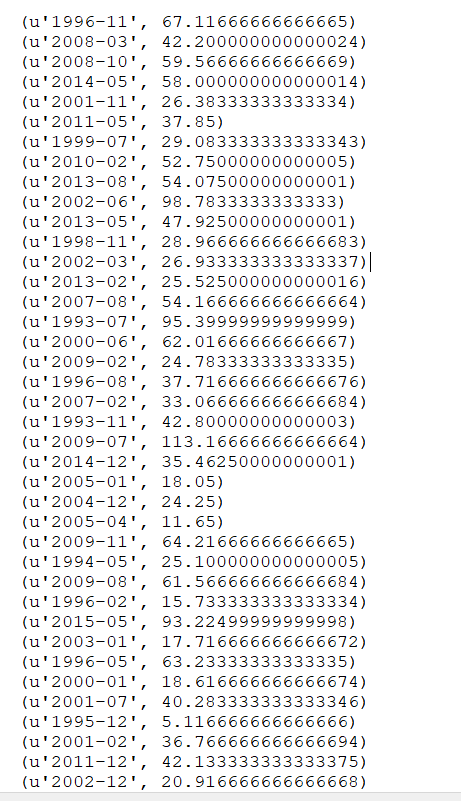
# Quesrion 4

# importing Spark Context  
from pyspark import SparkContext  
# Function to calculate maximum temperature and precipitation  
def max\_fun(a,b):  
 if a >= b:  
 return a  
 else:  
 return b  
# Function to calculate sum of daily precipitation   
def daily\_prec(a,b):  
 return a+b  
# adding spark context   
sc = SparkContext(appName = "lab1-q4")  
# reading temperature data  
temperature\_file = sc.textFile("/user/x\_samza/data/temperature-readings.csv")  
# reading precipitation data  
precipitation\_file = sc.textFile("/user/x\_samza/data/precipitation-readings.csv")  
# splitting temperature data into columns  
lines = temperature\_file.map(lambda line: line.split(";"))  
# forming a key value pair keeping station as key and temperature as value  
station\_temperature = lines.map(lambda x: (x[0],float(x[3])))  
# finding max temperature for each station  
max\_temperatures = station\_temperature.reduceByKey(max\_fun)  
# filtering max temperature between 25 and 30 degrees  
max\_temperatures = max\_temperatures.filter(lambda x: x[1] >= 25 and x[1] <= 30)  
# splitting precipitation data into columns  
lines\_prec = lines = precipitation\_file.map(lambda line: line.split(";"))  
# forming a key value pair keeping date and station as key and temperature as value  
station\_precipitation = lines\_prec.map(lambda x: ((x[0],x[1]),float(x[3])))  
# calculating sum of precipitation for each day for each station  
daily\_precipitation = station\_precipitation.reduceByKey(daily\_prec)  
# creating a key-value pair keeping station as key and daily precipitation as value  
daily\_precipitation = daily\_precipitation.map(lambda x: (x[0][0],float(x[1])))  
# calculating max daily precipitation for each station  
max\_precipitation\_station = daily\_precipitation.reduceByKey(max\_fun)  
# filtering max daily precipitation between 100 and 200 mm  
max\_precipitation\_station = max\_precipitation\_station.filter(lambda x: x[1] >= 100 and x[1] <= 200)  
# joining result of max temperature and max daily precipitation on the basis of stations  
max\_result = max\_temperatures.join(max\_precipitation\_station)  
# saving result  
max\_result.saveAsTextFile("prec\_temp\_station")

Because there is no matching between 2 Rdd, so result will be null.

# Quesrion 5

# importing required context  
from pyspark import SparkContext  
from operator import add  
   
sc = SparkContext(appName = "Lab1 Q5")  
#reading data  
ostergotland\_file = sc.textFile("/user/x\_samza/data/stations-Ostergotland.csv")  
#partition data  
stations = ostergotland\_file.map(lambda line: line.split(";"))  
map\_oster = stations.map(lambda x: (x[0]) )  
# collect and broadcast Stations  
stations = map\_oster.distinct().collect()  
Os\_stations = sc.broadcast(stations)  
precipitation\_file = sc.textFile("/user/x\_samza/data/precipitation-readings.csv")  
precipt\_lines = precipitation\_file.map(lambda line: line.split(";"))  
#filter stations  
filter\_precipt = precipt\_lines.filter(lambda x: x[0] in Os\_stations.value )  
#making key-value pair keeping year-month and stations as key and precipitation as value   
map\_precipt = filter\_precipt.map(lambda x: ((x[1][0:7],x[0]),float(x[3])))  
#calculating total monthly precipitation  
monthly\_prec = map\_precipt.reduceByKey(add)  
#find average and count  
avg\_prec = monthly\_prec.map(lambda x: ((x[0][0], (float(x[1]) ,1) )))  
result = avg\_prec.reduceByKey(lambda x,y: ( (x[0] + y[0]) , (x[1] + y[1]) ))  
result = result.map(lambda x: (x[0] , float(x[1][0] / x[1][1])))  
#partition and save data  
output = result.repartition(1)  
output.saveAsTextFile("lab1\_q5")



# Quesrion 6

from pyspark import SparkContext  
from operator import add  
sc = SparkContext(appName = "lab1 q6")  
staOstergotland\_file = sc.textFile("/user/x\_samza/data/stations-Ostergotland.csv")  
lines = staOstergotland\_file.map(lambda line: line.split(";"))  
#collect and broadcast  
stations = lines.map(lambda x: int(x[0]))  
stations = stations.distinct().collect() #collect to a python list  
Os\_stations = sc.broadcast(stations)  
#read temperature file  
temperature\_file = sc.textFile("/user/x\_samza/data/temperature-readings.csv")  
lines\_tempFile = temperature\_file.map(lambda line: line.split(";"))  
#Filter out Ostergoland stations  
temperatures = lines\_tempFile.filter(lambda x: (int(x[1][0:4]) >= 1950 and int(x[1][0:4]) <= 2014 and (x[0] in Os\_stations.value)))  
#monthly average temperature  
map\_temperature = temperatures.map(lambda x: ((x[1], int(x[0])), (float(x[3]), float(x[3]))))  
min\_max\_temp = map\_temperature.reduceByKey(lambda x, y:(min(x[0], y[0]), max(x[1], y[1])))  
min\_max\_temp = min\_max\_temp.map(lambda x: ((x[0][0][0:7], x[0][1]), (x[1][0]+x[1][1], 2)))  
min\_max\_temp = min\_max\_temp.reduceByKey(lambda x,y: (x[0]+y[0], x[1]+y[1]))  
average\_by\_month = min\_max\_temp.map(lambda x: ( (x[0][0]), (float(x[1][0]/ x[1][1]) , 1)))   
#year average   
#average\_by\_year = average\_by\_month.map(lambda x: (x[0], (x[1], 1)))  
average\_by\_year = average\_by\_month.reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))  
average\_by\_year = average\_by\_year.map(lambda x: (x[0], x[1][0] / x[1][1] ))  
#filter year 1950 to 1980  
average\_longTerm = average\_by\_year.filter(lambda x: (int(x[0][0:4]) >= 1950 and int(x[0][0:4]) <= 1980))  
average\_longTerm = average\_longTerm.map(lambda x: (x[0][5:7], (x[1], 1)))  
average\_longTerm = average\_longTerm.reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))  
average\_longTerm=average\_longTerm.map(lambda x: (x[0], x[1][0]/x[1][1] ))  
#collecting the result and map with average year  
results = average\_longTerm.collect()  
monthAvg = {month: temperature for (month, temperature) in results}  
difference\_temp = average\_by\_year.map(lambda x: (x[0], abs(x[1])-abs(monthAvg.get(x[0][5:7], 0))))  
#save result to output  
difference\_results = difference\_temp.repartition(1)\  
 .saveAsTextFile("lab1\_q6")

