Stat 243
Problem set 6

A. Strand, ID: 540441, GitHub: AndreasStrand

November 2, 2015

Problem 1

The code for problem 1 are included in code 1. The size of the original CSV is 12 Gb, the bzipped file is 1.7 Gb while the SQLite database is of size 9.4 Gb. The SQLite storage is in other words more efficiant than the .csv format, but less efficient than the bz2-zipped csv.

Listing 1: R code for problem 1 with comments.

```
setwd(”~/Documents/stat243”)

# Running this code bunch in the EC2
library(RSQLite)
year = seq(1987,2008, by =1)
airNames = paste(”http://www.stat.berkeley.edu/share/paciorek/”,
year, ”.csv.bz2”,sep = ””)
filename = ”airline.db”
db <- dbConnect(dbDriver(”SQLite”), dbname = filename)
for (i in 1:length(year)){
dbWriteTable(conn = db, name = ”airlines”,
              value = read.csv(bzfile(airNames[i]), header=T, sep = ”,”),
              row.names = FALSE, header = TRUE, append = T)
}
# Disconnecting using
dbDisconnect(db)
```

Problem 2

The answer for problem to is written as bash code with comments and pseudo code in code 2. Besides experimentation with the cluster, there is for some parts pseudo code to express the general idea. There was some error in my SQLite calculations in R, but for the speed comparison I would use system.time() in R and compared it to the equivalent for the pyspark RDD compations.

Listing 2: Bash code, pyspark code and pseudo code for problem 2 with comments.

```
setwd(”~/Documents/stat243”)
#### Logon to spark ####
export NUMBER_OF_WORKERS=12
cd ~/Documents/stat243/spark-1.5.1/ec2
export AWS_ACCESS_KEY_ID=‘grep aws_access_key_id ~/Documents/stat243/
stat243-fall-2015-credentials.boto | cut -d’ ’ -f3‘
export AWS_SECRET_ACCESS_KEY=‘grep aws_secret_access_key ~/Documents/stat243/
stat243-fall-2015-credentials.boto | cut -d’ ’ -f3‘
chmod 400 ~/Documents/stat243/.ssh/stat243-fall-2015-ssh_key.pem

./spark-ec2 -k andrstra@stud.ntnu.no:stat243-fall-2015 -i ~/Documents/
```

```
        stat243/.ssh/stat243-fall-2015-ssh_key.pem --region=us-west-2 -s
        ${NUMBER_OF_WORKERS} -v 1.5.1 launch sparkvm-andrstra@stud.ntnu.no
./spark-ec2 -k andrstra@stud.ntnu.no:stat243-fall-2015 -i ~/Documents
        /stat243/.ssh/stat243-fall-2015-ssh_key.pem --region=us-west-2 login sparkvm-
        andrstra@stud.ntnu.no
# terminating later using
./spark-ec2 --region=us-west-2 --delete-groups
        destroy sparkvm-andrstra@stud.ntnu.no


#### In spark ####
#### This section is containing pseudo code for problem 2
# Columns of airline data
 [1] "Year"              "Month"              "DayofMonth"
 [4] "DayOfWeek"         "DepTime"            "CRSDepTime"
 [7] "ArrTime"           "CRSArrTime"         "UniqueCarrier"
[10] "FlightNum"         "TailNum"            "ActualElapsedTime"
[13] "CRSElapsedTime"    "AirTime"            "ArrDelay"
[16] "DepDelay"          "Origin"             "Dest"
[19] "Distance"          "TaxiIn"             "TaxiOut"
[22] "Cancelled"         "CancellationCode"   "Diverted"
[25] "CarrierDelay"      "WeatherDelay"       "NASDelay"
[28] "SecurityDelay"     "LateAircraftDelay"

## Creating directories
export PATH=$PATH:/root/ephemeral-hdfs/bin/
hadoop fs -mkdir /data
hadoop fs -mkdir /data/airline
df -h
mkdir /mnt/airline

## Loading the airline data onto an HDFS
for i in `seq 1987 2008`;
do
        wget "http://www.stat.berkeley.edu/share/paciorek/$i.csv.bz2
done
hadoop␣fs␣-copyFromLocal␣/mnt/airline/*bz2␣/data/airline
#␣Check␣files␣on␣the␣HDFS
hadoop␣fs␣-ls␣/data/airline

##␣Installing␣numpy␣and␣running␣pyspark
yum␣install␣-y␣python27-pip␣python27-devel
pip-2.7␣install␣'numpy==1.9.2'
export␣PATH=$PATH:/root/spark/bin
pyspark
from␣operator␣import␣add
import␣numpy␣as␣np

##␣Creating␣an␣RDD␣"lines"
lines␣=␣sc.textFile('/data/airline')
numLines␣=␣lines.count()
def␣stratify(line)
␣␣␣␣vals␣=␣line.split(',')
␣␣␣␣return␣=␣(vals[16],␣1)
result␣=␣lines.map(stratify).reduceByKey(add).collect()
```

```
##␣Filter␣subset␣so␣it␣does␣not␣contain␣missing␣or␣unreasonable␣values␣for
#␣departure␣delay.␣Also␣repartitioning␣data␣such␣that␣it␣is␣equally␣spread␣across
#␣the␣twelve␣worker␣nodes
lines.filter(lambda␣line:␣(!NA␣>=0␣and␣<300)␣in␣line.split(',')[16]).repartition(12)

##␣Aggregating␣information␣to␣categories
#␣airline,␣departure␣airport,␣arrival␣airport,␣calendar␣month,␣day␣of␣week,
#␣hour␣of␣day␣of␣scheduled␣␣dep.␣time
#␣column␣numbers:␣11,␣17,␣18,␣2,␣4,␣5
#␣(for␣the␣last␣element␣take␣floor(<val␣5>/100)␣to␣get␣the␣hour␣of␣the␣sch.␣dep.time)
def␣computeKeyValue(line):
␣␣␣␣vals␣=␣line.split(',')
␣␣␣␣keyArr␣=␣[vals[x]␣for␣x␣in␣[11,17,18,2,4,5]]
␣␣␣␣keyArr[5]␣=␣floor(keyArr[5]/100)
␣␣␣␣#␣Creating␣key␣by␣joining␣column␣values␣by␣hyphen
␣␣␣␣keyVals␣=␣'-'.join(keyArr)
␣␣␣␣delay30␣=␣(vals[16]>=30)
␣␣␣␣delay60␣=␣(vals[16]>=60)
␣␣␣␣delay180␣=␣(vals[16]>=180)
␣␣␣␣return(keyVals,␣delay30,␣delay60,␣delay180)

#␣I␣am␣not␣sure␣about␣details␣in␣reduce␣by␣key,␣but␣the␣plan␣is␣that␣since
#␣delay30,␣delay60␣and␣delay180␣is␣boolean␣the␣average␣is␣the␣same␣as␣the␣share.
#␣Furthermore␣if␣the␣reduction␣for␣each␣key␣is␣recursive␣by␣the␣structure
#␣mean(mean(mean(a),b),c)␣I␣think␣it␣could␣work␣with␣the␣lambda␣suggested
avlambda␣=␣lambda␣v1,v2:␣numpy.average(v1,v2)
newRDD␣=␣lines.map(computeKeyValue).reduceByKey(avlambda)

##␣Saving␣the␣aggregated␣data␣set␣onto␣the␣master␣node
def␣strat(line):
␣␣␣␣changedLine␣=␣*change␣to␣comma-delimittid␣char.␣string*
␣␣␣␣return(changedLine)
newRDD.lines.map(strat).groupByKey().saveAsTextFile('/data/delayOverview')
```

Problem 3

The apply() family of functions in R are really useful, and I am looking forward to checking out the parallel apply calculations. I will look into it as we use in real work flows in class. I have to admit that I did not have the time now. By the way, what is the index from $Pr.2c$?

Problem 4

To extract certain columns from bash you may use code like the one below.

```
awk -F "\"*,\"*" '{print <column numbers>}' textfile.csv
```

This operation is fast, but even if it is done using piping ($unzip|*awk*|zip$) the zipping and unzipping will take some time. I do not think this is a worthwhile step.