Stat243 PS6

Meng Wang, SID: 21706745

October 31, 2015

1 Problem 1

First, use the bash command to download the data. We need to explicitly deal with the NA problem in the data. We could use the *awk* bash command to replace NA in the 16th column (departure time) with -99999, which is a number that does not happen in real world. We rename the files to year_ replace.csv in directory *ReplacedData*. The bash code is shown as below:

```
############## Problem 1 ####################
# download the file
url="http://www.stat.berkeley.edu/share/paciorek/1987-2008.csvs.tgz"
wget ${url}

# extract the file
tar -xvzf 1987-2008.csvs.tgz

# make the folder
mkdir ReplacedData

# replace the NA with -999
for filename in $(eval echo {1987..2008});
do
  echo "$filename"
  bzip2 -d "$filename".csv.bz2
  awk -F, '{ $16 = ($16 == "NA" ? -99999 : $16) } 1' OFS=, "./"$filename".csv" > "./ReplacedData/"$filen
done
```

Then we could use the *dbWriteTables* function for each year of the file and split the data to table. We name the table as *Table_Airline*. We also specify the column types of the table by setting the *colClasses* in function *dbWriteTable*. Here is the our code,

```
############# Problem 1(a) ###################
library(RSQLite)
db <- dbConnect(SQLite(), dbname="DB_Airline.sqlite")

dbWriteTable(conn = db, name = "Table_Airline", value = "./ReplacedData/1987_replaced.csv",
             row.names = FALSE, header = TRUE,
             colClasses = c(rep("numeric", 8), "factor", "numeric",
                            "factor", rep("numeric", 5), rep("factor", 2), rep("numeric", 4),
                            "factor", rep("numeric", 6)))
for(year in 1988:2008){
  dbWriteTable(conn = db, name = "Table_Airline", value = paste("./ReplacedData/",
                                                                year, "_replaced.csv", sep=""),
               row.names = FALSE, header = FALSE, append = TRUE,
               colClasses = c(rep("numeric", 8), "factor", "numeric",
                              "factor", rep("numeric", 5), rep("factor", 2), rep("numeric", 4),
                              "factor", rep("numeric", 6)))
}
# dbListTables(db)
# dbListFields(db, "Table_Airline")
# auth <- dbSendQuery(db, "select * from Table_Airline")
# fetch(auth, 5)
dbDisconnect(db)
```

The database file *DB_Airline.sqlite* is 11G, which is bigger than 1987-2008.csvs.tgz, which is 1.7G and smaller than the untared CSV file which is 12G.

2 Problem 2

2.1 (a)

First, let us do this in the SQLite.We select the rows which satisfy the condition that DepDelay is not NA. Then we use function *dbSendQuery* to create a view of the table. We save the filtered data as *Table_Airline_filter*. Here is the code,

```
library(RSQLite)
db <- dbConnect(SQLite(), dbname="DB_Airline.sqlite")
Table_Airline_filter <- dbSendQuery(db,
                                    "CREATE view Table_Airline_filter as
                                    SELECT * FROM Table_Airline
                                    WHERE DepDelay <> 'NA'")
```

Secondly, let us do this with Spark and Python code. It is very similar to the content of unit 7. We use function *filter* to get the subset and save the data to *airline_filter*. Here is the code for setting up

```
## setup HDFS
export PATH=$PATH:/root/ephemeral-hdfs/bin/
#HDFS mkdir
hadoop fs -mkdir /data
hadoop fs -mkdir /data/airline
#local mkdir
df -h
mkdir /mnt/airline
#cp files
scp wangmeng@184.23.19.240:/home/meng/Stat243/Homework/PS6/*bz2 /mnt/airline
#cp files from local to HDFS
hadoop fs -copyFromLocal /mnt/airline/*bz2 /data/airline
# check files on the HDFS, e.g.:
hadoop fs -ls /data/airline

## launch pyspark
# pyspark is in /root/spark/bin
export PATH=${PATH}:/root/spark/bin
# start Spark's Python interface as interactive session
pyspark
```

Here is the python code to run

```
## Stat243A, PS6 ###
# Meng Wang, SID: 21706745

##### Problem 2(a) #####
from operator import add
import numpy as np

#lines = sc.textFile('/data/airline/1987.csv.bz2').cache() # for testing
lines = sc.textFile('/data/airline').cache()
def screen(vals):
    vals = vals.split(',')
# 0 field is Year
# 15 field is DepDelay
# 18 field is Distance
# 3 field is DayOfWeek
    return(vals[0] != 'Year' and vals[15] != 'NA' and vals[18] != 'NA' and \
        vals[3] != 'NA')

lines = lines.filter(screen).repartition(192).cache()
lines.saveAsTextFile('/data/airline_filter')
```

2.2 (b)

First, let us use SQLite. Let us pick up one key *DayofWeek* and get the statistics for Arrival late more than 30 minutes.

```
## for Key DayofWeek
#proportion of flights more than 30 minutes late,
table_DayofWeek_30min_late <- dbGetQuery(db, "SELECT DayofWeek
                                    FROM Table_Airline_filter
                                    WHERE ArrDelay > 30" )
table(table_DayofWeek_30min_late)/nrow(table_DayofWeek_30min_late)
```

The answer gives

```
table_DayofWeek_30min_late
           0            1            2            3            4            5            6            7
1.383540e-06 1.432537e-01 1.378332e-01 1.472251e-01 1.640652e-01 1.708613e-01 1.056612e-01 1.310990e-01
```

Play the same game for different keys *ailine*, *arrivalairport*, *Dayofthemonth*, for different condition, i.e., late more than 30, 60, 180 minutes. It shall be very straight forward with the code above. Insert the key into the position after *SELETCT*. Insert the condition into the position after *WHERE* in the query. For example, the table for key DayofWeek with condition arrival late more than 60 minuets, could be computed by the following query.

```
table_DayofWeek_60min_late <- dbGetQuery(db, "SELECT DayofWeek
                                    FROM Table_Airline_filter
                                    WHERE ArrDelay > 60" )
```

Secondly, we use Spark for this job

```
from operator import add
import numpy as np
```

```
#lines = sc.textFile('/data/filter_1987').cache() # for testing
lines = sc.textFile('/data/airline_filter').cache()
def select_table(vals):
    vals = vals.split(',')
# 6 field is ArrDelay
# 3 field is DayOfWeek
    return(vals[6] >30 and vals[3] != 'NA')

lines_filter = lines.filter(select_table).repartition(192).cache()
```

2.3 (c)

This part of work has been done in part (b). The code to use in Spark is

```
lines = sc.textFile('/data/airline_filter').cache()
 ... define a filter ...
lines = lines.filter(filter).repartition(192).cache()
lines.saveAsTextFile('/data/airline_filter')
```

2.4 (d)

Here is the code to add an index

```
system.time(Indexed_Table <- dbGetQuery(db, "CREATE INDEX AirportID
                                        ON Table_Airline_filter(Origin)"))
# 356.821
```

Adding the index can greatly improve the speed of query. But creating the index takes a fair amount of time.

3 Problem 3

I use the package *foreach* and redo one of the query in 2(b) with 4 cores in my laptop. Here is the code,

```
library(foreach)
library(doParallel)
library(iterators)
library(RSQLite)
Fun <- function(){
  Avg <- dbSendQuery(db, "SELECT DayofWeek
                     FROM Table_Airline_filter
                     WHERE ArrDelay > 30")
  result <- fetch(Avg, -1)
  dbClearResult(Avg)
  return(result)
}

nCores <- 4
registerDoParallel(nCores)
out <- foreach(i = 1:100, .combine=rbind)%dopar%{
  outSub = Fun()
  outSub
}
```

This takes me 6296s in total, which is very slow.

4 Problem 4

For example, we want to cut the columns with flights that depart from SFO. Here is the bash code,

```
############ Problem 4 #####################
start=`date +%s`
awk -F',' '{
  if($17 == "SFO")
    print $0 > "subset_SFO"
}' ./*.csv
end=`date +%s`

runtime=$((end-start))
echo $runtime
# 530s
```

It takes me 530s, which is quite slow. So I will not suggest using it for pre-processing.
** R code saved in HW6.R; Python code saved in HW6.py; Bash codes saved in HW6.sh*