Homework 4

Stephanie Chan

March 7, 2013

Contents

1 Using Shell

Using grep, I have to find a way to get only the column that represents the

To find the number of flights per airport, I used a series of shell commands piped through. From this part of the project I learned some commands that

I hadn't used before. I used the command in R, originally intending to loop over each of the data files or the airport, until I thought of using `*` to get all the data files at once. The first part, I used a `cut` function to only get the origin airport. The next part of the pipe is `egrep '(LAX|OAK|SFO|SMF)'\`. The third part is a `sort` command over the set of origin airports, and the final piece was piped to `uniq -c` to count the number for each airport.

The total time for running this shell code was

```
user  system elapsed
144.261  19.905 324.771
```

Originally, I was concerned that the sort time would increase with polynomial time, so I considered counting the data files individually, and then adding the counts up in the end. I tried timing the counting with one/two/three files at a time, and found the total times to be 12.754, 29.388, 48.749. This increase in time was low enough to justify sorting all 20 files in one command, which ended up less than a multiple of 1.5 compared to the time for sorting each file.

The total counts for each of the four airports is

```
LAX  4057452
OAK  1151897
SFO  2711958
SMF   806133
```

2 Using R

In order to determine what the best size for reading in R would be, I checked a few different values of B where B is the number of lines read in each instance. These cases were determined using only the `2000.csv` data file.

value of B	time
1000	539.26
2000	436.51
5000	444.475
10000	477.938
20000	465.856
50000	438.534

It looks like the size of B does not have a significant impact on the time for these values of B that we tried.

I end up using R=50000 for each run. I didn't time this check, but it was significantly slower than using the shell or SQL, on the order of hours.

```
    LAX     OAK     SFO     SMF
4057452 1151897 2711958  806133
```

3 Using SQLite

To get the results, I created a data base. Creating the database and importing all the values took a long time. It took an especially long time to import all the values needed. Finding the counts using sqlite was pretty fast. It only took 52.8 seconds.

```
  Origin count(*)
1    LAX  4057452
2    OAK  1151897
3    SFO  2711958
4    SMF   806133
```

4 Computing the mean and standard deviation

4.1 Computing the mean and standard deviation with SQLite

I first tried to compute the mean and the standard deviation using SQLite. There was an average function in SQLite and I used that to get the results

```
  Origin avg(depdelay)
1    LAX      7.829167
2    OAK      7.183358
3    SFO      9.383507
4    SMF      7.041083
```

This query took 1048.6 seconds. A bit longer than I had hoped for. Unfortunately there was no standard deviation function available for SQLite. I attempted to find the standard deviation by computing the sum of squares and subtracting that by n*the average, but SQL returned negative numbers for the sum of squared. I sonder if that may be a function of the large numbers of square sums rolling over. This query took 970.2 seconds to run.

```
  Origin sum(depdelay*depdelay)
1    LAX            -1599222548
2    OAK              545265681
3    SFO            -1925995744
4    SMF              522824371
```

4.2 Computing the mean and standard deviation with R and the shell

Then I used the shell and R to find the mean and the standard deviation. The total timing for this was 441.9s. In this case, the necessary values to get were again the sum of all the departure delays, the sum of the squared departure delays, and the total number of counts. The means that were done through R matched the means from SQLite.

```
     LAX      OAK      SFO      SMF
7.829167 7.183358 9.383507 7.041083
```

The standard deviations are

```
     LAX      OAK      SFO      SMF
24.55805 20.53687 28.02638 24.47412
```

The sum of squares that was returned by R is

```
       LAX         OAK         SFO         SMF
2695744748  545265681 2368971552  522824371
```

The sum of squares returned the same values as SQL for OAK and SMF which are smaller values but the values are much bigger for LAX and SFO. It appears that the values roll over at 2^{32}.

5 Timing

The timings for each of the processes were done while there were other R processes and general computer tabs and windows open, so they may fluctuate based on how many other processes were used at that particular time.

5.1 Counting the number of rows

Shell	324.8
R	slow
SQL(import data)	slow
SQL query	52.8

SQL query was fastest, but it has a high initial cost to import all the files to the database. It would be most useful for instances where we have other uses for the database.

5.2 Getting the Average and Standard deviation.

The SQLite has an aggregator to get the Average, but unfortunately it did not have one for the standard deviation. Also, the size for the delays was too small to hold the sum of squares. This could possibly be fixed through changing the table schema.

Using R turned out to be more convenient especially because a lot of time was saved by using the shell to only pipe out the necessary lines.

SQL (get average)	1048.6
SQL (get squares)	970.2
R and shell	441.9

6 Appendix

6.1 counting with the shell

```
#!/bin/sh

DATAFILE=data/*.csv

cut -d, -f17 $DATAFILE | egrep '(LAX|OAK|SFO|SMF)' | sort | uniq -c

wc -l testcases/testcase.csv
```

6.2 counting using R (and some shell)

```
setwd("~/school/sta242/sta242hw4")

# set the shell
shell=system;
```

```
numlines = shell("wc -l testcase.csv",intern=TRUE)

# original test case
testcase = read.csv("testcases/testcase.csv",header=TRUE)
counts = shell("cut -d, -f17 testcases/testcase.csv | egrep '(LAX|OAK|SFO|SMF)' | sort
table(testcase[[17]])

# shell commands
# I also have a bash script that does this
datafile = "data/*.csv"
airportSearchTerm = "'(LAX|OAK|SFO|SMF)'"
shellCountAirportCommandBase = "cut -d, -f17 %s | egrep %s | sort | uniq -c"
shellCountAirportCommand = sprintf(shellCountAirportCommandBase, datafile, airportSearc

shellCountsOutput = shell(shellCountAirportCommand,intern=TRUE)

shelltime = system.time(shell(shellCountAirportCommand,intern=TRUE))
shelltime


# using R connection
con = file("data/2000.csv","r")
close(con)
B = 10000

originAirports=c("LAX","OAK","SFO","SMF")

getTotalUsingConnections = function(datafile,origin,B=10000) {
    print(datafile)
    con = file(datafile,"r")
    total = structure(integer(length(origin)),names=origin)
    while(TRUE) {
        ll = readLines(con,n=B)
        if(length(ll)==0)
          break
        tmp = sapply(strsplit(ll,","),`[[`,17)
        subCounts = table(tmp)[origin]
        subCounts[ is.na(subCounts) ] = 0
        total = total + subCounts
```

```
    }
    names(total)=origin
    close(con)
    return(total)
}

system.time(getTotalUsingConnections("data/2001.csv",originAirports,B=50000))

Sys.setlocale(locale="C")
alldata = sprintf("data/%s.csv",1987:2008)
totals = sapply(alldata,getTotalUsingConnections,origin=originAirports,B=50000)
```

6.3 using SQL

```
library(RSQLite)

dr = dbDriver("SQLite")

con = dbConnect(dr, dbname="airlineDelays.db")

## make it flexible to change
originAirports = c('LAX','OAK','SFO','SMF')
whereConditiontmp = sprintf("origin=='%s'",originAirports)
whereCondition = paste("where",paste(whereConditiontmp,collapse=" or "))


## get the counts from each departure airport
queryCounts = paste("SELECT origin, count(*) FROM delays", whereCondition, "group by or

r1 = dbSendQuery(con, queryCounts)
ans1 = fetch(r1)
print(ans1)

time1 = system.time(fetch(dbSendQuery(con, queryCounts)))

## get the average from each departure airport
queryAvg = paste("SELECT origin, avg(depdelay) FROM delays", whereCondition, "group by

r2 = dbSendQuery(con, queryAvg)
ans2 = fetch(r2)
```

```
print(ans2)

time2 = system.time(fetch(dbSendQuery(con, queryAvg)))


## get the variance of times from each departure airport
queryVar = paste("SELECT origin, sum(delays.depdelay*delays.depdelay) FROM delays", whe

r3 = dbSendQuery(con, queryVar)

ans3 = fetch(r3)
print(ans3)

time3 = system.time(fetch(dbSendQuery(con, queryVar)))


##
```

6.4 getting the average and standard deviation

```
shell = system


Sys.setlocale(locale="C")

getCountSumsSquares = function(year) {
    print(year)
    cmd = sprintf("cut -d, -f17,16 data/%s.csv | egrep '(LAX|OAK|SFO|SMF)'",year)
    con = pipe(cmd)

    B = 50000
    originAirports = c("LAX","OAK","SFO","SMF")
    records = matrix(0,3,4)

    counter=0
    ll = readLines(con)

    tmp = strsplit(ll,",")
    origin   = sapply(tmp, function(x) x[2])
    depdelay = sapply(tmp, function(x) as.numeric(x[1]))
```

```
    records = sapply(originAirports,
      function(x) {idx = as.logical(origin==x);
                   c(sum(idx), sum(depdelay[idx],na.rm=TRUE),sum(depdelay[idx]^2,na.rm=

    return(records)
}

time = system.time(getCountSumsSquares(2000))

years = 1987:2008
records = sapply(years,getCountSumsSquares)

totalRecords = rowSums(records)
## mean = sum/n
means = sapply(1:4, function(x) totalRecords[3*x-1]/totalRecords[3*x-2])
## standard deviation = sqrt((sumSquared - sum^2/n)/(n-1))
stdev = sapply(1:4, function(x)
       sqrt( (totalRecords[3*x] - totalRecords[3*x-1]^2/totalRecords[3*x-2]) / (totalRe
```