**Airline Data Streaming**

**Intro**

Data streaming has become an increasingly important technique in recent years due to both size and the dynamic nature of data. In this assignment, we work with data in “blocks”. In other words, we don’t read the entire data into memory but rather read a subset of the data, process it immediately, discard it right away and continue with the next subsets of the data. Therefore, we can have very large scale of data streaming into our memory and extract statistical information at the same time.

We will compare the speed of processing data using Unix shell tools, pure R tools and combination of both tools. We found that a connection between R and Unix is an efficient way to stream and process the data. Database tools such as SQLite is also explored to improve the efficiency of processing large scale of data. We will compute the number of flights leaving each of the airports (LAX, OAK, SFO, SMF) from 1987 to 2008. We will also compute the mean and standard deviation of the arrival delay times for flights departing from these four airports.

**Data Structure**

The data sets are constructed by 22 csv files (totally 12 Gigabytes), which contain domestic airline flights summaries as well as arrival and departure performance each year from 1987 to 2008. We have 29 columns in each data set for different years. We have particular interests in the 15th column for arrival delay (ARR\_DELAY) and 17th column for departure airport (ORIGIN). Other relevant time variable incudes YEAR, MONTH, DAY\_OF\_MONTH, DAY\_OF\_WEEK.

The data are available at <http://eeyore.ucdavis.edu/stat242/data/>. Please see Appendix I for R codes to download and uncompressed files programmatically.

**Tools Exploration**

The tools used in this assignment include Unix shell tools, R tools and database engine. We compare the speed of counting flights leaving each of the airports, specifically LAX, OAK, SFO and SMF for year 2008 to find the most efficient way to obtain statistical information (e.g. counts, means and standard deviations) for the entire data sets. Following is the system time needed to for counting:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Shell |  | R Streaming | |  | Database |
|  | **i. Multiple passes**  **(wc -l)** | **ii. One pass**  **(sort|uniq -c)** | **i. readLines**  **(file(csv))** | **ii. readLines**  **(pipe(egrep))** | | **RSQLite** |
| user | 67.933 | 62.077 | 268.294 | | 19.476 | 0.577 |
| system | 0.833 | 0.242 | 1.229 | | 0.061 | 0.016 |
| elapsed | 55.612 | 61.230 | 269.525 | | 57.478 | 0.576 |

Table -1. System.time for counts of flight leaving LAX, OAK, SFO and SMF in 2008

We found that Unix shell tools are extremely fast in taking subsets of data using “cut” and “grep” commands, which select the specific column and search with regular expression. “sort” and “uniq -c” is easier to write without doing multiple passes but it takes longer to sort given the data set is larger enough.

system.time(SHcounts.wc<-system("for airport in LAX OAK SFO SMF; do cut -f 17 -d , 2008.csv | grep $airport | wc -l; done",intern=TRUE))

system.time(SHcounts.uniq<-system("egrep '([0-9]|NA),(LAX|OAK|SFO|SMF),[A-Z]' 2008.csv | cut -f 17 -d , | sort | uniq -c",intern=TRUE))

Reading data in blocks directly from csv file in R is the slowest way to stream and process the data. However, prepare subset data in the Unix environment with regular expression significantly reduce the processing time from 269.525 seconds to 57.478 seconds. This is close enough to counting lines in shell scripting. Codes are available in Appendix II.

con<-pipe("egrep '([0-9]|NA),(LAX|OAK|SFO|SMF),[A-Z]' 2008.csv","r")

rcount<-function(B,con,year){

B=as.integer(B)

# set up counters

Rcounts<-structure(c(data=rep(0,4)),names=c("LAX","OAK","SFO","SMF"))

# reading blocks

while(TRUE){

txt=readLines(con,n=B)

if (length(txt)==0)

break

# counting

temp=sapply(strsplit(txt,","),"[[",17)

update<-as.numeric(table(temp)[names(Rcounts)])

update[is.na(update)]<-0

Rcounts<-Rcounts+update

}

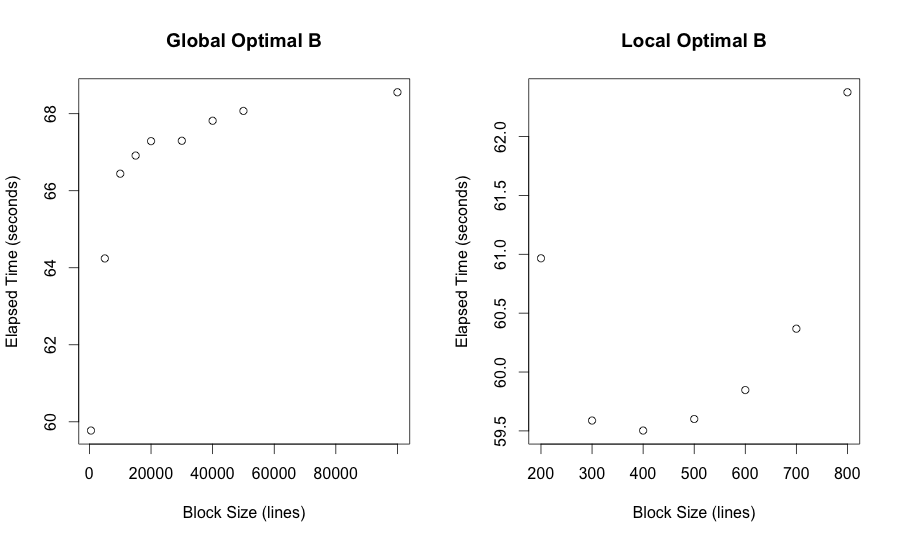
Rcounts

}

system.time(RSHcount<-rcount(400L,con,"2008"))

We are interested in exploring the best block size that is highly relevant to the performance for reading data while Unix is processing data line by line. The parameterized function and plot function in R are handy for us to find the optimal block size for streaming data from Unix to R. The optimal block size for reading lines directly from csv files and connection from Unix are different, specifically 12000L versus 4000L, because R is slower than Unix shell tools in taking the subsets of data and thus would be better of in dealing with larger block size in counting. But if we stream data indirectly from the connection of Unix using pipe(), we are more efficient in counting targeted observations with smaller block size than that in reading directly from csv files.

Following are plots for the exploration of optimal block size in the case of streaming prepared data from Unix to R :



The optimal B is 400 lines for an elapsed time less than 60 seconds if we pre-process data in Unix and then stream to R. We can expect to improve the efficiency by interfacing to C code because we could do faster while loops.

With the guarantee of speed, we would like to use R for its good extensibility with handy statistical tools. Functions such as table(), sum() and var() will help us to obtain the counts, mean and standard deviation.

**Statistics: counts, means and standard deviation**

Grounded on the data streaming in R reading from Unix, a function rstats() is developed to get the counts, obtain the sum and cumulate the variance for airlines leaving LAX, OAK, SFO and SMF. The mean is the sum divided by counts while the standard deviation is the square root of variance divided by counts.

*(I) Outliers*

Notice that the counts here are different from the counts obtained in the previous section (Tools Exploration) because we exclude the observations with NA on arrival delay. We will obtain more accurate mean and standard deviation for the arrival delay time by excluding the NA observations.

The counts of ORIGIN for each airport in 2008:

> LAX 184048; > OAK 53679; > SFO 118635; > SMF 45364; #NA included

> LAX 181308; > OAK 52818; > SFO 116029; > SMF 44907; #NA excluded

The counts of ORIGIN for each airport from 1987 to 2008:

> LAX 3947365; > OAK 1134520; > SFO 2605579; > SMF 790378; #NA included

> LAX 3879885; > OAK 1121028; > SFO 2551563; > SMF 782171; #NA excluded

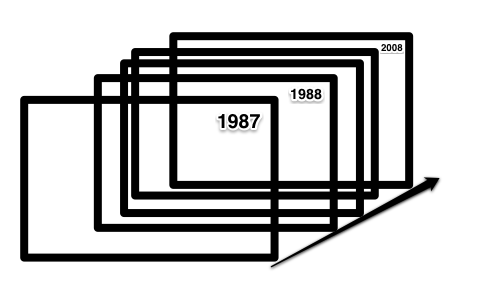
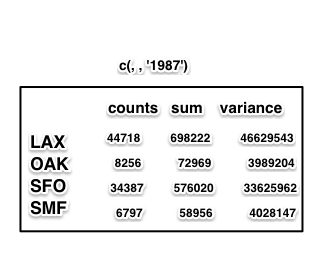
*(II) AWK Validation*

We use awk with under UNIX to do a quick validation of the sum of arrival delay time. It would print the counts and cumulated arrival delay time. The statistical results in R are validated by this AWK validation for 1987, 2001 and 2008.

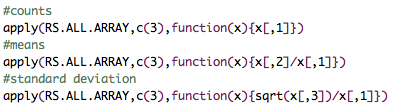
system.time(cmean<-system("egrep '([0-9]|NA),LAX,[A-Z]' 2008.csv | cut -f 15 -d , | awk 'BEGIN {s=0; c=0}; {s=s+$1;c=c+1}; END {print c,s, s/c}'",intern=TRUE))

*(III) Results*

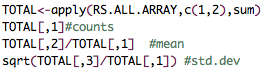
The statistical results are constructed as arrays. Columns are the statistics including counts, sum, variance; rows are the airports of our interests, specifically for LAX, OAK, SFO and SMF; the third dimension is the year from 1987 to 2008. Please see Appendix III for the statistics obtained from data streaming.



We can use apply function to obtain the mean and std.dev for each year. Results are available in Appendix IV.



To obtain the statistics for the entire data set from 1987 to 2008, we can use apply function to sum up the counts, sum and variance through out the time.



The statistical results for the entire data set from 1987 to 2008 are shown in the table below. Statistics for individual year are available in Appendix V and VI.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Counts | Means | Std.Dev |
| LAX | 3879885 | 6.011921 | 26.31361 |
| OAK | 1121028 | 5.027054 | 21.09526 |
| SFO | 255156 | 8.008065 | 29.06354 |
| SMF | 782171 | 5.342205 | 24.14323 |

**Conclusion**

In this assignment, we work with data in “blocks” to break through the limitation of memory and improve speed of data processing. Learning data streaming is an important step to learn parallel computing which carry out the calculations simultaneously. If data streaming is processing data in an time series way, parallel computing is processing data from an cross sectional prospective.

Besides the idea of reading data in blocks, I learn how to use Unix shell tools to pre-process data and open connections via pipe() to stream the data into R. Basic commands such as grep, cut, uniq, sort and the redirection operators in the shell are very handy in pre-processing data. In R, I learn how to open connections include pipe, file, url, bzfile, gzfile, xzfile.

Computing the same information from relational database is part of the assignment. I create tables and write queries in SQLite. It turns out that the speed is amazingly fast. Database makes life easier with command such as GROUP BY (similar to tapply).

Character encoding is an issue while dealing with the csv file for 2001 and 2002. It is important to read the data with correct encoding which pairs each character from a given repertoire particular bit patterns.

URL for the repositiory on GitHub:

https://github.com/ykangxie/airport.git

**Appendix I.**

##############################################

# Download and Unzip Files

##############################################

#/\*get fileURl\*/

#txt = readLines("http://eeyore.ucdavis.edu/stat242/data")

#ll = grep("csv.bz2", txt) #line.number containing "csv.bz2"

#files = gsub(".\*([0-9]{4}.(csv).bz2).\*", "\\1", txt[ll]) #file names

#fileUrl<-sprintf("http://eeyore.ucdavis.edu/stat242/data/%s", files) #fileUrl

#/\*download bz2 files\*/

#destFile<-sprintf("./Desktop/airport/%s",files)

#mapply(function(url,output){download.file(url,destfile=output,method="curl")},fileUrl,destFile)

#/\*unzip bz2 files\*/

#cmd<-sprintf("cd ./Desktop/airport; bunzip2 -d %s", files)

#sapply(cmd,system)

**Appendix II. Tools Exploration**

files = gsub(".\*([0-9]{4}.csv).\*", "\\1", list.files()[grep(".\*([0-9]{4}.csv).\*", list.files())])

year = gsub(".\*([0-9]{4}).csv.\*", "\\1", files[grep(".\*([0-9]{4}.csv).\*", files)])

######################################################

# Tools Exploration- system.time(counts) for 2008.csv

######################################################

setwd("./Desktop/airport")

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

year="2008"

**#(I) Shell Scripting**

**#1.1 /grep & wc -l/ [multiple passes]**

system.time(SHcounts.wc<-system("for airport in LAX OAK SFO SMF; do cut -f 17 -d , 2008.csv | grep $airport | wc -l; done",intern=TRUE))

#user system elapsed

#67.933 0.833 55.612

**#1.2 /egrep & (sort + uniq -c)/ [one pass]**

system.time(SHcounts.uniq<-system("egrep '([0-9]|NA),(LAX|OAK|SFO|SMF),[A-Z]' 2008.csv | cut -f 17 -d , | sort | uniq -c",intern=TRUE))

#user system elapsed

#62.077 0.242 61.230

**#(II) R Streaming (csv files)**

rstreaming<-function(B,csvfile="2008.csv"){

B=as.integer(B)

con=file(csvfile,"r")

Rcounts<-structure(c(data=rep(0,4)),names=c("LAX","OAK","SFO","SMF"))

while(TRUE){

txt=readLines(con,n=B)

if (length(txt)==0)

break

temp=sapply(strsplit(txt,","),"[[",17)

update<-as.numeric(table(temp)[names(Rcounts)])

update[is.na(update)]<-0

Rcounts<-Rcounts+update

}

Rcounts

}

system.time(rstreaming(400L,"2008.csv")) #268.294 1.229 269.525

system.time(rstreaming(12000L,"2008.csv")) #255.774 0.816 256.571

**#(III) R Streaming from Unix via pipe()**

con<-pipe("egrep '([0-9]|NA),(LAX|OAK|SFO|SMF),[A-Z]' 2008.csv","r")

rcount<-function(B,con,year){

B=as.integer(B)

Rcounts<-structure(c(data=rep(0,4)),names=c("LAX","OAK","SFO","SMF"))

while(TRUE){

txt=readLines(con,n=B)

if (length(txt)==0)

break

temp=sapply(strsplit(txt,","),"[[",17)

update<-as.numeric(table(temp)[names(Rcounts)])

update[is.na(update)]<-0

Rcounts<-Rcounts+update

}

Rcounts

}

system.time(RSHcount<-rcount(400L,con,"2008"))

#user system elapsed

#19.476 0.061 57.478

**Appendix III. Function rstats()**



**Appendix IV. Codes for All-Years Statistics**

##############################################

# All Years: 1987-2012.csv

##############################################

source('rstats.R')

Sys.setlocale(locale="C")

**# i. Each Year (array)**

year = gsub(".\*([0-9]{4}).csv.\*", "\\1", files[grep(".\*([0-9]{4}.csv).\*", files)])

cmds<-sprintf("egrep '([0-9]|NA),(LAX|OAK|SFO|SMF),[A-Z]' %s.csv",year)

system.time(RS.ALL<-lapply(cmds,function(cmd){con<-pipe(cmd,"r")

tmp<-rstats(400L,con)

close(con)

tmp}))

#user system elapsed #1824.276 10.011 1285.353

names(RS.ALL)<-year

RS.ALL.ARRAY<-array(unlist(RS.ALL), dim = c(nrow(RS.ALL[[1]]), ncol(RS.ALL[[1]]), length(RS.ALL))) #reconstruction: (4 x 3 x 22)

rownames(RS.ALL.ARRAY)<-rownames(RS.ALL[[1]])

colnames(RS.ALL.ARRAY)<-colnames(RS.ALL[[1]])

dimnames(RS.ALL.ARRAY)[[3]]<-year

#counts

apply(RS.ALL.ARRAY,c(3),function(x){x[,1]})

#means

apply(RS.ALL.ARRAY,c(3),function(x){x[,2]/x[,1]})

#standard deviation

apply(RS.ALL.ARRAY,c(3),function(x){sqrt(x[,3])/x[,1]})

**# ii. All Year (matrix)**

TOTAL<-apply(RS.ALL.ARRAY,c(1,2),sum)

TOTAL[,1]#counts

TOTAL[,2]/TOTAL[,1] #mean

sqrt(TOTAL[,3]/TOTAL[,1]) #std.dev

TOTAL.STATS<-structure(data.frame(cbind(TOTAL[,1],TOTAL[,2]/TOTAL[,1],sqrt(TOTAL[,3]/TOTAL[,1]))),names=c("counts", "means", "std.dev"))

**# iii. All Years (awk-validation)**

# notes: the awk validation is a quick validation which has a larger n because of the inclusion of NA ArrDelay observations

system.time(cmean2<-system("egrep '([0-9]|NA),LAX,[A-Z]' [12]\*.csv | cut -f 15 -d , | awk 'BEGIN {s=0; c=0}; {s=s+$1;c=c+1}; END {print c,s,s/c}'",intern=TRUE))

# counts, sum & means for (LAX [12]\*.csv)

#[1] "3947365 5.90915"

**Appendix V. Statistical Information for Each Year**

> RS.ALL.ARRAY

, , 1987

Rcounts Rsum Rsumvar

LAX 44718 698222 46629543

OAK 8256 72969 3989204

SFO 34387 576020 33625962

SMF 6797 58956 4028147

, , 1988

Rcounts Rsum Rsumvar

LAX 168315 494824 72032151

OAK 28069 119662 8219810

SFO 130309 987464 66265631

SMF 27202 77202 8481110

, , 1989

Rcounts Rsum Rsumvar

LAX 161229 1026141 82462028

OAK 27849 170309 9736698

SFO 124400 909651 75711416

SMF 25118 136822 11395138

, , 1990

Rcounts Rsum Rsumvar

LAX 168100 1017515 77791138

OAK 36207 125261 10410748

SFO 128921 820036 61602123

SMF 24919 107973 8888563

, , 1991

Rcounts Rsum Rsumvar

LAX 155002 1059089 80226036

OAK 39793 159760 11079901

SFO 122423 1098101 66316359

SMF 26805 117460 8886301

, , 1992

Rcounts Rsum Rsumvar

LAX 153756 668930 61829138

OAK 36497 55264 6543787

SFO 121240 550297 44178952

SMF 29709 69666 8026645

, , 1993

Rcounts Rsum Rsumvar

LAX 151020 445402 58514989

OAK 39316 82182 7731066

SFO 116523 411976 49284859

SMF 29460 89367 8990148

, , 1994

Rcounts Rsum Rsumvar

LAX 151919 706131 56519438

OAK 44138 182071 9094462

SFO 117053 528507 48494829

SMF 28717 123118 8317834

, , 1995

Rcounts Rsum Rsumvar

LAX 175894 1674907 117273324

OAK 64524 364986 22515629

SFO 126283 1128107 86908794

SMF 35628 176701 15460096

, , 1996

Rcounts Rsum Rsumvar

LAX 179193 1840182 144844249

OAK 59669 346971 22738325

SFO 132074 1639385 138390756

SMF 36424 230301 22428645

, , 1997

Rcounts Rsum Rsumvar

LAX 183447 1496839 131055064

OAK 57718 302076 19093393

SFO 135199 1371023 106307297

SMF 36767 229068 18147255

, , 1998

Rcounts Rsum Rsumvar

LAX 178421 1145846 153113710

OAK 55057 364247 27315707

SFO 133557 1748564 181440746

SMF 36073 252998 21311474

, , 1999

Rcounts Rsum Rsumvar

LAX 185812 1612380 156361588

OAK 54725 366694 22596092

SFO 131444 1206069 153878008

SMF 36308 261399 20734114

, , 2000

Rcounts Rsum Rsumvar

LAX 203598 2430501 237508835

OAK 56106 616442 40836851

SFO 127532 1857824 200120593

SMF 38301 421899 31972521

, , 2001

Rcounts Rsum Rsumvar

LAX 124679 649956 82336267

OAK 47206 319414 22143188

SFO 26523 80109 22390373

SMF 26216 148242 13099860

, , 2002

Rcounts Rsum Rsumvar

LAX 170178 44932 102203954

OAK 59085 278342 27347972

SFO 86901 40047 61185023

SMF 37570 140758 20426443

, , 2003

Rcounts Rsum Rsumvar

LAX 221122 169021 126010691

OAK 66662 99080 29496751

SFO 120866 93931 70251106

SMF 46338 114649 23423413

, , 2004

Rcounts Rsum Rsumvar

LAX 229731 980717 154611681

OAK 70049 320588 38459632

SFO 127990 526323 88707721

SMF 48110 328413 37680364

, , 2005

Rcounts Rsum Rsumvar

LAX 228122 1081568 153685230

OAK 69652 377980 38562684

SFO 127055 813418 118866372

SMF 50161 314357 35258171

, , 2006

Rcounts Rsum Rsumvar

LAX 230381 1341989 195359125

OAK 73761 383264 49994165

SFO 129245 1234712 143417625

SMF 53218 265041 41288074

, , 2007

Rcounts Rsum Rsumvar

LAX 233940 1779392 221358053

OAK 73871 419482 43786892

SFO 135609 1582832 172304007

SMF 57423 364578 51483842

, , 2008

Rcounts Rsum Rsumvar

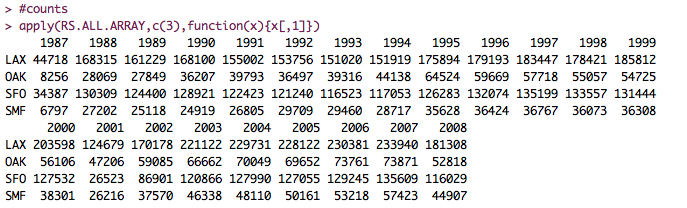
LAX 181308 961077 174730165

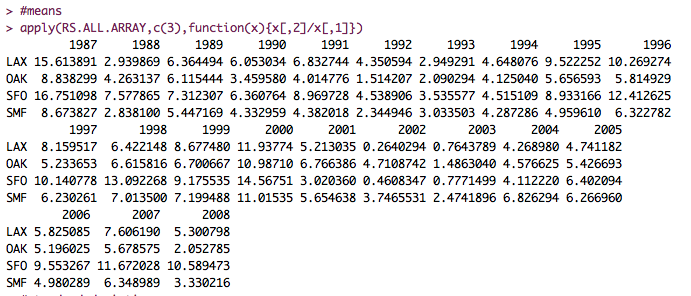
OAK 52818 108424 27175568

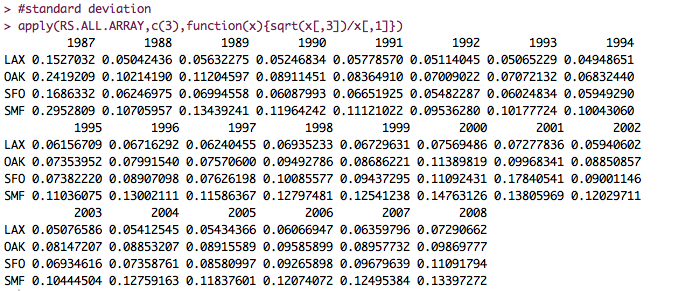
SFO 116029 1228686 165629128

SMF 44907 149550 36196022

**Appendix VI. Statistical Results for Each Year**







**Appendix VII. SQL Part**  
#----------------------------

#SQLite

#----------------------------

library("RSQLite")

dr=dbDriver("SQLite")

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

system.time(rr<-dbSendQuery(con,"SELECT count(\*) FROM delays WHERE Origin IN ('LAX', 'OAK', 'SFO', 'SMF') GROUP BY Origin;"))

system.time(fetch(rr,100))

#counts

fetch(rr,100)

# user system elapsed

# 0.557 0.016 0.576

library("RSQLite.extfuns")

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

init\_extensions(con)

#average

aa<-dbSendQuery(con,"SELECT avg(ArrDelay) FROM delays WHERE Origin IN ('LAX', 'OAK', 'SFO', 'SMF') GROUP BY Origin;")

fetch(aa,4)

#std.dev

std<-dbSendQuery(con,"SELECT stdev(ArrDelay) FROM delays WHERE Origin IN ('LAX', 'OAK', 'SFO', 'SMF') GROUP BY Origin;")

fetch(std,4)