# STAT 243: Problem Set 6

Jin Rou New [jrnew]

November 1, 2014

## 1 Problem 1

The database file is 9.56 Gb, smaller than the original CSV (12 Gb) but much larger than the bzipped copy (1.2Gb) and the .ffData binary representation (1.3 Gb).

```
library(RSQLite)
data_dir <- "data"
procdata_dir <- "procdata"
years <- 1987:2008

# Download all bz2 files
for (year in years) {
  data_filepath <- paste0(year,".csv.bz2")
  if (!file.exists(file.path(data_dir, data_filepath)))
    download.file(paste0("http://www.stat.berkeley.edu/share/paciorek/", data_filepath),
                  file.path(data_dir, data_filepath))
}

# Read in bz2 files, convert NA values and write out to SQL database
db <- dbConnect(SQLite(), dbname = file.path(procdata_dir, "flights.db"))
col_classes <- c("integer", rep("factor", 3), rep("integer", 4),
                 "factor", "integer", "factor", rep("integer", 5), rep("factor", 2),
                 rep("integer", 4), "factor", rep("integer", 6))
for (year in years) {
  is_first_file <- year == years[1]
  data_filepath <- paste0(year,".csv.bz2")
  con <- bzfile(file.path(data_dir, data_filepath), "rt")
  data <- read.csv(con, header = TRUE, colClasses = col_classes)
  data$DepDelay[is.na(data$DepDelay)] <- -9999
  dbWriteTable(conn = db, name = "flights", value = data,
               row.names = FALSE, append = !is_first_file)
}
dbDisconnect(db)
closeAllConnections()
```

## 2 Problem 2

(a) and (b) Results are displayed below. Oddly, subsetting is faster for ff but calculating the mean by group is faster for SQLite. Calculating the median by group too 484s for ff, but could not be done for SQLite because the required `RSQLite.extfuns` package is not available for `R 3.0.2` on the server. Code for doing the same on Spark is given below, even though I could not get it to run without strange errors on the server.

```
# ff
# Subset
  user  system elapsed
32.162   5.796  81.773
# Mean
   user   system  elapsed
488.909  293.738 2892.788
# Median
   user   system  elapsed
484.602  239.916 2067.493

# SQLite
# Subset
  user  system elapsed
52.644   7.408  60.304
# Mean
   user  system elapsed
222.101  28.813 258.769
```

```
# With Spark
import numpy as np
import time
from operator import add
lines = sc.textFile("/data/airline").repartition(96)

def subset(line):
  vals = line.split(',')
  return(vals[0] != 'Year' and vals[15] != 'NA' and
  float(vals[15]) >= -30 and float(vals[15]) <= 720)

# Subset flights data
lines_filtered = lines.filter(subset).collect()
lines_filtered.take(5) # Check

# Extract flights departing from SFO or OAK
print "%f " %  time.time()
lines_bayarea = lines_filtered.filter(lambda line: any(airport in line.split(',')[16]
  for airport in ('SFO', 'OAK')))
lines_bayarea.take(5)
print "%f " %  time.time()

# Find mean/median departure delay by airport
print "%f " %  time.time()
def depdelay(line):
  vals = line.split(',')
  if vals[0] == 'Year':
    return('0', 0)
  else:
    return(vals[16], float(vals[15]))

def getmean(input):
  if len(input) == 2:
    if len(input[1]) > 0:
```

```
      m = np.mean(input[1])
      return((input[0], m))
    else:
      return((input[0], -999))
  else:
    return((input[0], -9999))

def getmedian(input):
  if len(input) == 2:
    if len(input[1]) > 0:
      m = np.median(input[1])
      return((input[0], m))
    else:
      return((input[0], -999))
  else:
    return((input[0], -9999))

lines_depdelay = lines.map(lines_depdelay)
depdelay_mean = lines_depdelay.groupByKey().map(getmean).collect()
depdelay_median = lines_depdelay.groupByKey().map(getmedian).collect()
print "%f " %  time.time()
```

```
library(RSQLite.extfuns)
# With SQLite
db <- dbConnect(SQLite(), dbname = file.path(procdata_dir, "flights.db"))

# Subset flights data
if (!("flights_subset" %in% dbListTables(db))) {
  viewquery <- paste0("create view flights_subset as select * from ",
                      "flights where DepDelay >= -30 and DepDelay <= 720")
  dbGetQuery(db, viewquery)
}

# Extract flights departing from SFO or OAK
system.time({
  query1 <- "select * from flights_subset where Origin = 'SFO' or Origin = 'OAK'"
  flights_bayarea <- dbGetQuery(db, query1)
})

# Find mean/median departure delay by airport
system.time({
  query2 <- "select Origin, avg(DepDelay) as meanDepDelay from flights_subset group by Origin"
  depdelay_mean <- dbGetQuery(db, query2)
})

# Notes: Requires RSQLite.extfuns which is not available for R 3.0.2 on AWS
# init_extensions(db)
# system.time({
#   query3 <- "select Origin, median(DepDelay) as medianDepDelay from flights_subset group by Origin"
#   depdelay_median <- dbGetQuery(db, query3)
# })

dbDisconnect(db)
```

```
library(ff)
library(ffbase)
library(doBy)
library(data.table)

# With ff
# Download all ff files
data_filepaths <- c("AirlineDataAll.ffData", "AirlineDataAll.RData")
for (data_filepath in data_filepaths) {
  if (!file.exists(file.path(data_dir, data_filepath)))
    download.file(paste0("http://www.stat.berkeley.edu/share/paciorek/", data_filepath),
                  file.path(data_dir, data_filepath))
}
ffload(file.path(data_dir, "AirlineDataAll"), rootpath = "/mnt/airline")

# Subset flights data
select <- dat$DepDelay >= -30 & dat$DepDelay <= 720 & !is.na(dat$DepDelay)
indices_select <- ffwhich(select, select == TRUE)
dat_subset <- dat[indices_select, ]

# Extract flights departing from SFO or OAK
# Note: is.element() and %in% do not work with ffvectors
system.time({
  select_bayarea <- dat_subset$Origin == "SFO" | dat_subset$Origin == "OAK"
  indices_select_bayarea <- ffwhich(select_bayarea, select_bayarea == TRUE)
  flights_bayarea <- dat_subset[indices_select_bayarea, ]
})

# Find mean/median departure delay by airport
system.time({
  depdelay_mean <- ffdfdply(x = dat_subset, split = as.character(dat_subset$Origin),
                            FUN = function(x) {
                              dt <- data.table(x)
                              dt[, list(meanDepDelay = mean(DepDelay)), by = Origin]
                            })
})
system.time({
  depdelay_median <- ffdfdply(x = dat_subset, split = as.character(dat_subset$Origin),
                              FUN = function(x) {
                                dt <- data.table(x)
                                # Note: Need to convert to double else error
                                # when an integer is returned
                                dt[, list(medianDepDelay = as.double(median(DepDelay))),
                                  by = Origin]
                              })
})
```

(c) I added an index on the departure airport using `create index` before rerunning the code from parts (a) and (b). Results are shown below. Adding the index clearly improves speed.

```
# SQLite
# Subset
  user  system elapsed
14.317   1.659  30.918
```

```
# Mean
   user  system elapsed
138.967  33.410 314.509
```

```
# Add index to Origin
if (!file.exists(file.path(procdata_dir, "flights-indexed.db")))
  file.copy(file.path(procdata_dir, "flights.db"), file.path(procdata_dir, "flights-indexed.db"))
db_indexed <- dbConnect(SQLite(), dbname = file.path(procdata_dir, "flights-indexed.db"))
indexquery <- "create index OriginID on flights(Origin)"
dbGetQuery(db_indexed, indexquery)

if (!("flights_subset" %in% dbListTables(db_indexed))) {
  # Subset flights data
  viewquery <- paste0("create view flights_subset as select * from ",
                      "flights where DepDelay >= -30 and DepDelay <= 720")
  dbGetQuery(db_indexed, viewquery)
}

# Extract flights departing from SFO or OAK
system.time({
  query1 <- "select * from flights_subset where Origin = 'SFO' or Origin = 'OAK'"
  flights_bayarea2 <- dbGetQuery(db_indexed, query1)
})

# Find mean/median departure delay by airport
system.time({
  query2 <- "select Origin, avg(DepDelay) as meanDepDelay from flights_subset group by Origin"
  depdelay_mean2 <- dbGetQuery(db_indexed, query2)
})
dbDisconnect(db_indexed)
```

(d) I first extracted a list of unique airports before doing a `foreach` loop over the airports.
I added an index on the departure airport using `create index` before rerunning the code from parts (a) and (b). Results are shown below. Parallelisation vastly improves the speed of the operation.

```
user  system elapsed
148.439  30.706  86.888
```

```
library(foreach)
library(doParallel)
library(iterators)
# Find mean departure delay by airport (in parallel)
db <- dbConnect(SQLite(), dbname = file.path(procdata_dir, "flights-indexed.db"))
num_cores <- 4
registerDoParallel(num_cores)
query <- "select Origin from flights_subset"
airports <- unique(dbGetQuery(db, query)[[1]])
system.time({
  depdelay_mean3 <- foreach(airport = airports, .combine = rbind) %dopar% {
    db <- dbConnect(SQLite(), dbname = file.path(procdata_dir, "flights.db"))
    query <- paste0("select Origin, avg(DepDelay) as meanDepDelay ",
                    "from flights_subset where Origin = '", airport ,"'")
    mean_temp <- dbGetQuery(db, query)
```

```
  }
})
dbDisconnect(db)
```

# 3 Problem 3

(a) Results are shown below.

```
   user  system elapsed
614.633  36.145 720.723
```

```
# With SQLite
db <- dbConnect(SQLite(), dbname = file.path(procdata_dir, "flights.db"))
# Extract the 20 observations with the longest departure delays
# from airports with at least 1 million flights
system.time({
  viewquery <- paste("create view num_departing_flights as select Origin,",
                     "count(Origin) as NumDepartures from flights group by Origin")
  dbGetQuery(db, viewquery)
  joinquery <- paste0("create view flights_all as select * from flights_subset ",
                      "join num_departing_flights on flights_subset.Origin = ",
                      "num_departing_flights.Origin")
  dbGetQuery(db, joinquery)
  searchquery <- paste0("select * from flights_all2 where NumDepartures >= 1000000 ",
                        "order by DepDelay desc limit 20")
  dep_delays_longest <- dbGetQuery(db, searchquery)
})
dbDisconnect(db)
```

(b) To avoid sorting large tables, I obtained a permutation vector of the sorted (in descending order) departure delay vector and collected the indices of that vector with ranks 1 to 20 (for 20 longest departure delays), then used the 20 indices to extract the relevant observation
Note: Code could not run on full data set on server, despite running successfully on a subset locally and on the server, so results on speed of operations are not available.

```
# With ff
ffload(file.path(data_dir, "AirlineDataAll"), rootpath = "/mnt/airline")

# Subset flights data
select <- dat$DepDelay >= -30 & dat$DepDelay <= 720 & !is.na(dat$DepDelay)
indices_select <- ffwhich(select, select == TRUE)
dat_subset <- dat[indices_select, ]

# Extract the 20 observations with the longest departure delays
# from airports with at least 1 million flights
system.time({
  num_departing_flights <- ffdfdply(x = dat, split = as.character(dat$Origin),
                                    FUN = function(x)
                                      summaryBy(Origin ~ Origin, data = x,
                                                FUN = sum, keep.names = FALSE))
  dat_merged <- merge(dat_subset, num_departing_flights, by = "Origin")
  select_num_departures <- dat_merged$Origin.sum > 1000000
```

```
  indices_select_num_departures <- ffwhich(select_num_departures,
                                           select_num_departures == TRUE)
  dat_merged_subset <- dat_merged[indices_select_num_departures, ]
  select_top20 <- fforder(dat_merged_subset$DepDelay, decreasing = TRUE) <= 20
  indices_top20 <- ffwhich(select_top20, select_top20 == TRUE)
  dat_top20 <- dat_merged_subset[indices_top20, ]
  dat_top20 <- dat_top20[order(dat_top20$DepDelay, decreasing = TRUE), ]
})
```

(c) To avoid sorting large tables, I extracted a subset of the table with departure delays of more than 700 min before sorting the resulting smaller table.
Note: Code could not run on pyspark, so results on speed of operations are not available.

```
# With Spark
num_departing_flights = lines_filtered.map(count).reduceByKey(add).collect()
lines_merged = lines_filtered.join(num_departing_flights).collect()

# Extract the 20 observations with the longest departure delays from
# airports with at least 1 million flights
def getlongestdelays(line):
  vals = line.split(',')
return(int(vals[29]) > 100000 and float(vals[15]) > 700)

lines_delays_temp = lines_filtered.filter(getlongestdelays)
lines_longest_delays = lines_delays_temp.sortByKey(ascending = False,
                                                   keyfunc = lambda line: float(line.split(',')[15])).ta
print "%f " %  time.time()
```

# 4 Problem 4

The operation took 529s, slower than both SQLite and ff.

```
files_bz=$(ls data | grep bz2)
for file_bz in $files_bz
do
bzip2 -dk data/$file_bz
done

echo "Start: " > procdata/unix-systime.txt
date +"%s" >> procdata/unix-systime.txt
files=$(ls data | grep csv)
for file in $files
do
cat data/$file | egrep ",(SFO|OAK),[[:alpha:]]"
done
echo "End: " >> procdata/unix-systime.txt
date +"%s" >> procdata/unix-systime.txt
```