Stat243: Problem Set 6

Worked with Jamie Palumbo, Mingyung Kim, Alanna Iverson

Sicun Huang

November 2, 2015

1. Set up EC2 virtual machine.

```
#login
ssh -i ~/.ssh/stat243-fall-2015-ssh_key.pem ubuntu@52.32.71.69
mkdir -p mnt/airline
cd mnt
#increse storage in /mnt
git clone "https://github.com/berkeley-stat243/stat243-fall-2015"
cd stat243-fall-2015/howtos
sudo ./setup-storage
cd ~/mnt/airline
#download airline dataset
wget http://www.stat.berkeley.edu/share/paciorek/1987-2008.csvs.tgz
tar -xvzf 1987-2008.csvs.tgz

#start R
export PATH=${PATH}:/root/R/bin
R
```

Create an SQLite database from R with a single table to hold the airline data. To save memory, use the individual year files to build up table in pieces. Because RSQLite converts NAs to zeros in numeric fields, replace the missing values in the fields with a numeric code. Here we choose 9999. Then, examine size of the database.

```
install.packages("RSQLite")
library(RSQLite)
setwd("/home/ubuntu/mnt/airline")

drv <- dbDriver( "SQLite" )
#connect to database
db <- dbConnect( drv, dbname = "airline" )

#set up database
#read in data from first year
dat <- read.csv("1987.csv.bz2")
#substitute all NA's for 9999
dat[is.na(dat)] <- 9999
#write to database
dbWriteTable(conn=db, name="airline", value=dat, row.names=FALSE)
rm(dat)

#do the same for the other years, append to the same table
```

```r
years <- seq( 1988, 2008 )
for (i in years){
  val <- paste0( i, ".csv.bz2" )
  dat <- read.csv( file=val )
  dat[is.na(dat)] <- 9999
  dbWriteTable(conn = db, name = "airline", value = dat, row.names = FALSE, append= TRUE)
  rm(dat)
}

#check database size
size <- file.size("airline")
utils:::format.object_size(size, "Gb")
#[1] "10.3 Gb"
```

Observe that database size is smaller than original csvs (12 Gb) but bigger than the bzipped copy of the original csvs (1.7 Gb).

2. First do it for SQLite.

(a) Subset database in SQLite to omit flights with missing values for the departure delay or values that seem unreasonable.

```r
cleandb <- dbConnect( drv, dbname = "cleanairline" )
#only include columns we will need later in the problem
#create new column for hour of day of the scheduled departure time
data <- dbGetQuery(db, "Select Year, Month, DayOfMonth, DayOfWeek, DepTime, CRSDepTime,
                   ArrTime, CRSArrTime, UniqueCarrier, FlightNum, ArrDelay, DepDelay,
                   Origin, Dest, Distance, (CRSDepTime-(CRSDepTime%100))/100 as CRSDepHour
                   FROM airline
                   WHERE DepDelay < 9999 and ActualElapsedTime > 0 and CRSElapsedTime > 0")
dbWriteTable(conn = cleandb, name = "cleanairline", value = data, row.names = FALSE)
rm(data)
```

(b) Compare speed at which Spark and SQLite aggregate information into categories where there is one category for each unique combination of airline, departure airport, arrival airport, calendar month, day of week, and hour of day of the scheduled departure time. For each category (i.e. key), compute proportion of flights more than 30 minutes, 60 minutes, and 180 minutes late.

```r
queryFun <- function( conn, query ) {
  df <- dbGetQuery( conn, query )
  return(df)
}

#query creates new columns DepDelay30, DepDelay60, DepDelay180 for flights more than 30,
#60, 180 minutes late and new column total for total number of flights; compute proportions
#of flights more than 30, 60, 180 minutes late; group data by the specified categories
query <- "SELECT COUNT(*) as total,
SUM(CASE WHEN DepDelay>30 THEN 1 ELSE 0 END) as DepDelay30,
SUM(CASE WHEN DepDelay>60 THEN 1 ELSE 0 END) as DepDelay60,
SUM(CASE WHEN DepDelay>180 THEN 1 ELSE 0 END) as DepDelay180,
CAST(SUM(CASE WHEN DepDelay>30 THEN 1 ELSE 0 END) AS FLOAT)/CAST(COUNT(*) AS FLOAT) AS prop30,
CAST(SUM(CASE WHEN DepDelay>60 THEN 1 ELSE 0 END) AS FLOAT)/CAST(COUNT(*) AS FLOAT) AS prop60,
CAST(SUM(CASE WHEN DepDelay>180 THEN 1 ELSE 0 END) AS FLOAT)/CAST(COUNT(*) AS FLOAT) AS prop180,
UniqueCarrier,Origin,Dest,Month,DayOfWeek,CRSDepHour
FROM cleanairline
GROUP BY UniqueCarrier,Origin,Dest,Month,DayOfWeek,CRSDepHour"
```

```
system.time( queryFun( cleandb, query ) )
# user  system elapsed
# 614.428  33.716 659.848
```

(d) Add index to SQLite database, repeat (b) and compare time.

```
#create index
dbGetQuery(cleandb, "CREATE INDEX ind ON cleanairline(UniqueCarrier,Origin,Dest,Month,DayOfWeek,
           CRSDepHour)")

system.time( queryFun(cleandb,query) )
# user  system elapsed
# 225.620  27.560 253.183
```

The processing time is greatly reduced after adding index.

(e) Report the top 10 groupings (keys) in terms of proportion of late flights for groupings with at least 150 flights.

```
df <- queryFun(cleandb,query)
#subset keys with at least 150 flights
data <- subset(df,total>=150)
head( data[order(-data$prop30),],10 )
#         total DepDelay30 DepDelay60 DepDelay180    prop30    prop60     prop180
# 6517747   160         66         28           0 0.4125000 0.1750000 0.000000000
# 6582556   151         61         18           1 0.4039735 0.1192053 0.006622517
# 6517520   150         57         31           3 0.3800000 0.2066667 0.020000000
# 6517748   152         57         22           0 0.3750000 0.1447368 0.000000000
# 6583038   163         60         25           0 0.3680982 0.1533742 0.000000000
# 6517407   158         58         19           1 0.3670886 0.1202532 0.006329114
# 5192821   162         59         36           1 0.3641975 0.2222222 0.006172840
# 6583522   172         62         25           6 0.3604651 0.1453488 0.034883721
# 6518208   165         58         20           3 0.3515152 0.1212121 0.018181818
# 6517745   177         62         28           1 0.3502825 0.1581921 0.005649718
#         UniqueCarrier Origin Dest Month DayOfWeek CRSDepHour
# 6517747            WN    DAL  HOU     6         5         20
# 6582556            WN    HOU  DAL     2         5         19
# 6517520            WN    DAL  HOU     4         5         20
# 6517748            WN    DAL  HOU     6         5         21
# 6583038            WN    HOU  DAL     6         5         19
# 6517407            WN    DAL  HOU     3         5         20
# 5192821            UA    LAX  SFO    12         5         11
# 6583522            WN    HOU  DAL    10         5         19
# 6518208            WN    DAL  HOU    10         5         20
# 6517745            WN    DAL  HOU     6         5         18
```

2. Then we setup Spark. Download the airline dataset to the master node and load the individual .bz2 files onto the HDFS.

```
#setup Spark cluster
#create UNIX environment variables containing AWS credentials
export AWS_ACCESS_KEY_ID=`grep aws_access_key_id stat243-fall-2015-credentials.boto | cut \
-d' ' -f3`
export AWS_SECRET_ACCESS_KEY=`grep aws_secret_access_key stat243-fall-2015-credentials.boto \
```

```
| cut -d' ' -f3`
#change permissions on the private SSH key file
chmod 400 ~/.ssh/stat243-fall-2015-ssh_key.pem
cd spark-1.5.1/ec2
export NUMBER_OF_WORKERS=12
#start a spark cluster
./spark-ec2 -k sc.huang@berkeley.edu:stat243-fall-2015 -i ~/.ssh/stat243-fall-2015-ssh_key.pem \
--region=us-west-2 -s ${NUMBER_OF_WORKERS} -v 1.5.1 launch sparkvm-sc.huang
#login
./spark-ec2 -k sc.huang@berkeley.edu:stat243-fall-2015 -i ~/.ssh/stat243-fall-2015-ssh_key.pem \
--region=us-west-2 login sparkvm-sc.huang

mkdir /mnt/airline
cd /mnt/airline
#download airline dataset to master node
wget http://www.stat.berkeley.edu/share/paciorek/1987-2008.csvs.tgz
tar -xvzf 1987-2008.csvs.tgz

export PATH=$PATH:/root/ephemeral-hdfs/bin/
#set up directories in HDFS
hadoop fs -mkdir /data/airline
#copy the dataset onto it
hadoop fs -copyFromLocal /mnt/airline/*bz2 /data/airline
hadoop fs -ls /data/airline

#install python packages
yum install -y python27-pip python27-devel
pip-2.7 install 'numpy==1.9.2'
#start python in spark
export PATH=${PATH}:/root/spark/bin
pyspark
```

Read the data into a Spark RDD; repartition so the dataset is equitably spread across the worker nodes in Spark cluster.

```
from operator import add
import numpy as np
#read data into Spark from the HDFS
lines = sc.textFile('/data/airline')
#repartition the dataset to better distribute data across the nodes
lines = lines.repartition(192).cache()
```

(a) Filter datasets in Spark to omit flights with missing values for the departure delay or values that seem unreasonable.

```
filterLines=lines.filter(lambda line: 'NA' not in line.split(',')[15]).cache()
```

(b) Repeat (b) from above for spark. Compare time.

```
#write map function to create key for each unique combination and set up for later \
#computations of total number of flights as well as number of flights that are more \
#than 30 minutes, 60 minutes, and 180 minutes late for each unique key
def mapper( line ):
  vals = line.split(',')
```

```
  #convert scheduled departure time into whole hours
  schDep = list( str(vals[5]) )
  if len(schDep)==4:
    hour = ''.join(schDep[0:2])
  elif len(schDep)==3:
    hour = schDep[0]
  else:
    hour = str(0)
  #create key for each unique combination
  key = '-'.join([vals[x] for x in [8,16,17,1,3]])+'-'+hour
  #count used to calculate total number of flights
  total=1.0
  #initialize counts for flights more than 30 minutes, 60 minutes, and 180 minutes late
  valid1=0
  valid2=0
  valid3=0
  #if flight more than 30 minutes, 60 minutes, or 180 minutes late, set count to 1
  if vals[15]>30:
    valid1 = 1.0
  if vals[15]>60:
    valid2 = 1.0
  if vals[15]>180:
    valid3 = 1.0
  return(key, (total,valid1,valid2,valid3))

#reduce function to aggregate statistic; get total number of flights and number of flights \
#that are more than 30 minutes, 60 minutes, and 180 minutes late for each unique key
def reducer( (a, b, c, d), (w, x, y, z) ):
  return( (a+w),(b+x),(c+y),(d+z) )

import time
#function to time aggregation operation
def timeAggr( line ):
  start = time.time()
  #apply map function to RDD
  mappedLines = line.map(mapper)
  #apply reduce function to resulting RDD; returns a RDD
  tmp = mappedLines.reduceByKey(reducer)
  #collect to make RDD accessible
  results = tmp.collect()
  elapsed = time.time()-start
  return(tmp,results,elapsed)

tmp,results,elapsed = timeAggr(filterLines)

tmp
#PythonRDD[11] at collect at <stdin>:5

#results has form (key,(total,valid1,valid2,valid3))
results[0:3]
#[(u'NW-STL-MSP-3-3-15', (38.0, 38.0, 38.0, 38.0)), (u'DL-ATL-SNA-11-4-19', (10.0, 10.0, \
#10.0, 10.0)), (u'CO-GSO-MCO-8-6-15', (4.0, 4.0, 4.0, 4.0))]
```

```
elapsed
#500.448312997818
```

The above operations are slightly faster in Spark.
(c) Get the resulting aggregated dataset onto the disk of the master node as a single data file.

```
#use repartition(1) to get a single data file instead of multiple pieces from the different \
#nodes hosting the HDFS
tmp.repartition(1).saveAsTextFile('/data/airline')

exit()
```

```
hadoop fs -ls /data/airline
#Warning: $HADOOP_HOME is deprecated.

#Found 2 items
#-rw-r--r--   3 root supergroup          0 2015-10-29 04:58 /data/airline/_SUCCESS
#-rw-r--r--   3 root supergroup  351289530 2015-10-29 04:56 /data/airline/part-00000

#copy file to master node
hadoop fs -copyToLocal /data/airline/part* /mnt/airline

head part*
#(u'DH-CVG-MDW-12-5-21', (4.0, 4.0, 4.0, 4.0))
#(u'DL-SLC-SMF-5-5-22', (27.0, 27.0, 27.0, 27.0))
#(u'OO-MKE-CLE-9-6-7', (4.0, 4.0, 4.0, 4.0))
#(u'OH-ORF-CVG-3-5-13', (4.0, 4.0, 4.0, 4.0))
#(u'US-JAX-CLT-9-4-10', (61.0, 61.0, 61.0, 61.0))
#(u'WN-BUR-PHX-4-4-11', (41.0, 41.0, 41.0, 41.0))
#(u'EV-ORD-CVG-7-3-13', (3.0, 3.0, 3.0, 3.0))
#(u'HP-SEA-ANC-9-4-11', (4.0, 4.0, 4.0, 4.0))
#(u'OH-CVG-BNA-4-4-7', (4.0, 4.0, 4.0, 4.0))
#(u'NW-MEM-OKC-6-6-14', (38.0, 38.0, 38.0, 38.0))
```

Terminate spark cluster.

```
./spark-ec2 --region=us-west-2 --delete-groups destroy sparkvm-sc.huang
```

3. Use parallel apply function to compute the proportion of delayed flights by key using the SQLite database. Here separate tasks by month. Compare time to the other approaches used above.

```
require(parallel)
require(doParallel)

#set number of cores
registerDoParallel(4)

taskFun <- function(i) {
  #open separate database connections for the separate tasks for them to operate in parallel
  newdb <- dbConnect( SQLite(), dbname = "cleanairline" )
  query <- paste0("SELECT COUNT(*) as total,
  SUM(CASE WHEN DepDelay>30 THEN 1 ELSE 0 END) as DepDelay30,
  SUM(CASE WHEN DepDelay>60 THEN 1 ELSE 0 END) as DepDelay60,
```

```
  SUM(CASE WHEN DepDelay>180 THEN 1 ELSE 0 END) as DepDelay180,
  CAST(SUM(CASE WHEN DepDelay>30 THEN 1 ELSE 0 END) AS FLOAT)/CAST(COUNT(*) AS FLOAT) AS prop30,
  CAST(SUM(CASE WHEN DepDelay>60 THEN 1 ELSE 0 END) AS FLOAT)/CAST(COUNT(*) AS FLOAT) AS prop60,
  CAST(SUM(CASE WHEN DepDelay>180 THEN 1 ELSE 0 END) AS FLOAT)/CAST(COUNT(*) AS FLOAT) AS prop180,
  UniqueCarrier,Origin,Dest,Month,DayOfWeek,CRSDepHour
  FROM cleanairline WHERE Month=",i,
  " GROUP BY UniqueCarrier,Origin,Dest,Month,DayOfWeek,CRSDepHour", sep="")
  df <- dbGetQuery(newdb,query)
  }

system.time(dfPar <- mclapply(1:12, taskFun, mc.cores=4))
# user   system  elapsed
# 1070.076   84.436  326.106

head(dfPar[[2]])
#   total DepDelay30 DepDelay60 DepDelay180    prop30    prop60 prop180
# 1     7          0          0           0 0.0000000 0.0000000   0.000
# 2     7          1          0           0 0.1428571 0.0000000   0.000
# 3     7          0          0           0 0.0000000 0.0000000   0.000
# 4     7          2          2           0 0.2857143 0.2857143   0.000
# 5     8          2          1           1 0.2500000 0.1250000   0.125
# 6     8          2          1           0 0.2500000 0.1250000   0.000
#   UniqueCarrier Origin Dest Month DayOfWeek CRSDepHour
# 1            9E    ABE  DTW     2         1          6
# 2            9E    ABE  DTW     2         1         12
# 3            9E    ABE  DTW     2         1         16
# 4            9E    ABE  DTW     2         2          6
# 5            9E    ABE  DTW     2         2         12
# 6            9E    ABE  DTW     2         2         16
```

Parallelization with indexing is faster than both SQLite and Spark without indexing but slower than SQLite with indexing.

4. Use bash tools to cut out the columns not needed without explicitly unzipping the files.

```
col=1,2,3,4,5,6,7,8,9,10,15,16,17,18,19

time for ((i=1987; i<=2008; i++))
do
bzcat $i.csv.bz2 | cut -d',' -f${col}| bzip2 > clean$i.csv.bz2
done
# real 16m39.767s
# user 20m12.716s
# sys 0m26.456s
```

The whole process took 17 minutes. It helps to reduce file size hence reduce time needed to read in data during database construction.