RSS Matters

Research and Statistical Support
University of North Texas

Using R packages "ff" and "filehash" to Handle Big Data

Link to the last RSS article here: Un-modeled confounders: Don't get burned by Simpson's Paradox. -- Ed.

By Dr. Rich Herrington, Research and Statistical Support Consultant

This month we take a look at the R packages "ff", "filehash" and "biglm" as fairly straightforward approaches to handling large data sets in R (>1 GB). The CRAN URL's for these packages can be found at

- http://cran.r-project.org/web/packages/filehash/index.html
- http://cran.r-project.org/web/packages/ff/index.html
- http://cran.r-project.org/web/packages/biglm/index.html

The default behaviour of R is to create and store all objects (data, R scripts, functions, etc.) within RAM (working memory). Within R, the function "memory.size()" displays the amount of RAM being used by R at the memory.size query time. The function "gc()" (garbage can collection) will recover allocated memory and provide additional information about R memory usage.

A critical difference between the two packages is how objects are dealt with as they are referenced - "filehash" will load the referenced objects into RAM only when they are actually needed for a calculation, whereas ff will load "pieces of an object" as the pieces are needed (or, these pieces are being paged in and out of memory as they are needed). Potentially, package "ff" can handle much larger data sets more efficiently without depleting RAM. The goal of package "filehash" is to only work with the complete object pieces (i.e. variables) rather than an entire collection of objects (i.e. a dataframe or list). Practically I have found both packages to work fairly well with moderately large data files (approx. 1 GB), with a slight edge favoring package "ff".

The data set that is used to illustrate these two packages is a simulated dataset created by my collegue Jon Starkweather - found at:

http://www.unt.edu/rss/class/Jon/Example_Sim/

The dataset is approximately 890 MB with 62 columns (variables) and 1,023,027 rows. I purposely ran this demonstration on a fairly modest computer setup - an older computer with the following specs: Intel Pentium 4 (1 x CPU) 3.20 GHz; 2.9 GB available RAM; running MS Windows XP Professional SP3.

Additionally, we use a modified "ls()" function script posted at "StackOverFlow":

http://stackoverflow.com/questions/1358003/tricks-to-manage-the-available-memory-in-an-r-session

Additionally, we have posted this script at:

http://www.unt.edu/rss/rich/ls.objects.r

This script will need to be placed in the working R directory so that it can be # read in at the beginning of the session. This function "lsos()" will provide more # readable output concerning object sizes and their memory resources, than the stock# "ls()" function.

We first demonstrate the "ff" package:

Click here to see the program code. Note: Comments are preceeded by "#"

Next month, we'll look at how "ff" and "filehash" perform with analysis methods that are known to use large amounts of memory e.g. clustering, tree partitioning, bootstrap model selection, bayesian model selection, etc.

Attachment	Size
RSS.pdf	24.47 KB

Contact Us:
University Information Technology
1155 Union Circle #310709
Denton, TX 76203 USA
Voice: 940-565-4068
Fax: 940-565-4060

Visit Us:

Sage Hall, Room 338
http://it.unt.edu/benchmarks/

Email us:
Have questions on content or technical issues? Please contact us.
unt.uit@unt.edu

```
# Set working directory; read in the R function "lsos"; and
# load packages "ff", "ffbase", "filehash" and "biglm":

setwd("d:\\rich\\Rdata")
source("ls.objects.r")

require(ff)
require(ffbase)
require(filehash)
require(biglm)

# Memory "garbage collection"
gc()
memory.size()
lsos()

# We start out with about 29.58 MB of memory allocated to the R
# session:

# > gc()
#
# used (Mb) gc trigger (Mb) max used (Mb)
# Ncells 749885 20.1    1166886 31.2   899071 24.1
# Vcells 631899  4.9    1162592  8.9  1031040  7.9
#
# > memory.size()
# [1] 29.58
#
# > lsos()
# Type  Size Rows Columns
# lsos function 2,900   NA      NA


# The function "read.csv.ffdf" will read the large comma delimited text
# by chunks of 1000 records.  It is important to leave the "colClasses"
# parameter undefined so that no assumptions are made about what the
# data classes are (e.g. interger, double, factor, ordered, character, etc.).

system.time(data.x <-
            read.csv.ffdf(file="E:\\Examplonia_Population_Data_19Jun2012.txt",
                          header=TRUE, colClasses=NA, first.rows=1000))

# > system.time(data.x <-
#   +             read.csv.ffdf(file="E:\\Examplonia_Population_Data_19Jun2012.txt",
#   +                           header=TRUE, colClasses=NA, first.rows=1000))
# user  system elapsed
# 557.34   10.25  686.99
# > 686.99/60
# [1] 11.44983
#
#
#
# The file input took approximately 11 minutes.  Previous attempts to read the file
```

```
# in lead to out of memory conditions.  Examination of the memory utilization shows
# an increase of RAM allocation but is recovered by invoking "gc()":

lsos()
memory.size()
gc()
lsos()
memory.size()

# > lsos()
# Type    Size    Rows Columns
# data.x     ffdf 131,784 1012027      62
# lsos   function   2,900      NA      NA
#
# > memory.size()
# [1] 169.03
#
# > gc()
# used (Mb) gc trigger  (Mb) max used  (Mb)
# Ncells  788251 21.1    2380321  63.6  2959476  79.1
# Vcells 1684463 12.9   16459353 125.6 20574192 157.0
#
# > lsos()
# Type    Size    Rows Columns
# data.x     ffdf 131,784 1012027      62
# lsos   function   2,900      NA      NA
#
# > memory.size()
# [1] 50.33

# We now have an "ff" object - "data.x" - that resides on physical disk,
# for which the internal R representation requires 2,900 Bytes of RAM.

# Examining the variable names:

names(data.x)

# > names(data.x)
# [1] "id"                  "region"              "city.names"          "gender"
# [5] "age"                 "education"           "income"              "q1.cognitive.1"
# [9] "q2.cognitive.2"      "q3.cognitive.3"      "q4.cognitive.4"      "q5.physical.1"
# [13] "q6.physical.2"      "q7.physical.3"       "q8.physical.4"       "q9.physical.5"
# [17] "q10.social.1"       "q11.social.2"        "q12.social.3"        "q13.social.4"
# [21] "q14.social.5"       "neuroticism"         "extroversion"        "openness"
# [25] "conscientiousness"  "agreeableness"       "nuclear"             "coal"
# [29] "nat.gas.electric"   "wind"                "solar"               "automobile"
# [33] "bus"                "train"               "bicycle"             "walk"
# [37] "gasoline"           "nat.gas.car"         "hybrid"              "electric"
# [41] "other"              "animal.extinction"   "plant.extinction"    "severe.storms"
# [45] "ice.melt"           "sea.rise"            "religion"            "abortion"
# [49] "national.debt"      "unemployment"        "healthcare"          "public.edu"
# [53] "campaign.finance"   "business.regulation" "social.responsibility" "tobacco.user"
# [57] "blood.type"         "bmi"                 "sys.bp"              "wbc.count"
# [61] "glucose"            "ldl.cholesterol"
```

```
# Save and reload "ff" objects; Creates a file on hard drive that is approx. 300 MB;
# The original data file was approx. 883 MB.

ffsave(data.x,file="Examplonia.ff")

# Useful information concerning the filesystem names, paths, etc.
ffinfo("Examplonia.ff")

# If needed, this function reloads the ff object
ffload("Examplonia.ff")


gc()
memory.size()
lsos()

# At this point we are using about 46 MB of RAM.

# > gc()
#
# used (Mb) gc trigger  (Mb) max used  (Mb)
# Ncells  790447 21.2    2380321  63.6  2959476  79.1
# Vcells 1687670 12.9   13167482 100.5 20574192 157.0
#
# > memory.size()
# [1] 45.93
#
# > lsos()
# Type             Size      Rows     Columns
# data.x     ffdf   131,784   1012027  62
# lsos   function     2,900   NA       NA


# Package "ff" can work with the package "biglm" to perform memory
# bound linear model fitting - both linear and generalized linear models.

# Function "bigglm" from package "ffbase"
system.time(lm.fit.1<-bigglm(income ~ gender, data=data.x))
lm.fit.1

lsos()
memory.size()
gc()
lsos()
memory.size()

# Fitting this linear model - linearly predicting income from gender status
# does not appreciably increase the amount of memory allocated to R.

# > lsos()
# Type             Size    Rows        Columns
# data.x       ffdf 131,784 1012027      62
# lm.fit.1    bigglm  19,956      13      NA
```

```
# lsos     function   2,900      NA      NA
#
# > memory.size()
# [1] 86.88
#
# > gc()
# used (Mb) gc trigger (Mb) max used  (Mb)
# Ncells  805510 21.6    2380321 63.6  2959476  79.1
# Vcells 1699018 13.0    6741749 51.5 20574192 157.0
#
# > lsos()
# Type             Size    Rows    Columns
# data.x       ffdf 131,784 1012027      62
# lm.fit.1    bigglm  19,956      13      NA
# lsos     function   2,900      NA      NA
#
# > memory.size()
# [1] 46.57

summary(lm.fit.1)

# > summary(lm.fit.1)
# Large data regression model: bigglm(income ~ gender, data = data.x)
# Sample size =  1012027
# Coef       (95%        CI)      SE      p
# (Intercept)    61328.6958 61212.3239 61445.0678 58.1860 0.0000
# gender[T.male]    29.7456  -136.2981   195.7894 83.0219 0.7201

rm(lm.fit.1)

# The total time to run this model was approximately 10.34 seconds.

# > system.time(lm.fit.1<-bigglm(income ~ gender, data=data.x))
# opening ff C:/DOCUME~1/RICHHE~1.UNT/LOCALS~1/Temp/RtmpclE7QQ/ffdfa2c6489231f.ff
# user  system elapsed
# 9.75    0.08   10.34

# Some stock functions (e.g. mean, hist, plot) will work with the "ff" (by utilizing the
# accompanying "ffbase" package), however some functions are specifically tailored, and
# named such, to work with package "ff" (e.g. "table.ff"):

hist(data.x$income)
mean(data.x$income)

# > mean(data.x$income)
# [1] 61343.31

table.ff(data.x$gender, useNA="no")

# > table.ff(data.x$gender, useNA="no")
# opening ff C:/DOCUME~1/RICHHE~1.UNT/LOCALS~1/Temp/RtmpclE7QQ/ffdfa2c347c2b59.ff
#
# female   male
# 514927 497100
```

```
#
# > ls()
# [1] "data.x"

lsos()
memory.size()
gc()
lsos()
memory.size()

# An examination of the allocated memory (also invoking "gc"), brings the total memory
# back to approximately 47 M.

# > lsos()
#
# Type             Size    Rows        Columns
# data.x       ffdf 131,784 1012027      62
# lm.fit.1   bigglm  19,956     13      NA
# lsos     function   2,900      NA      NA
#
# > memory.size()
# [1] 77.55
#
# > gc()
# used (Mb) gc trigger (Mb) max used  (Mb)
# Ncells  812904 21.8    2380321 63.6  2959476  79.1
# Vcells 1706569 13.1    6110221 46.7 20574192 157.0
#
# > lsos()
# Type             Size   Rows        Columns
# data.x       ffdf 131,784 1012027      62
# lm.fit.1   bigglm  19,956     13      NA
# lsos     function   2,900      NA      NA
#
# > memory.size()
# [1] 46.65

# Off loading data objects to physical memory while recovering allocated
# memory with "gc" seems to be a promisiung way of handeling the ever
# problematic memory depletion "creep".


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


# We know turn our attention to the "filehash" package.  In many ways
# the filehash package is easier to use than package "ff".  Once a
# file hash database has been created, we can use all of the familiar
# indexing operators and functions that R provides to start with:
# e.g. "$", A[i,j], min, max, lm, hist).

# Run only the first time to create an initial empty filehash database;
# You do not need to create this db again - only a "dbInit" is needed.
# initialize. A "dbCreate" will write over previous saved versions of the db.
```

```
dbCreate("Examplonia.db")

# Initialize db for reading and writing
db <- dbInit("Examplonia.db")

# We have used the "ff" package to read in the comma delimited text file,
# and we use the following "for loop" below to read the data into a
# filehash database on physical disk.
# Three commands are necessary:  "dbCreate", "dbInsert", and "dbInit".
# "dbCreate" is used to initialize an empty database;  "dbInit" assigns
# the reference to this database into the working session (here we assign it
# to "db").  From that point forward, normal indexing can be used to extract
# data and assign data to the "db" object.  Assignments to "db" will update
# the physical database on physical disk.

# Insert data table (ff objects at this point) into the filehash db

for(i in 1:ncol(data.x)){
 dbInsert(db, colnames(data.x)[i], data.x[,i])
}

ls()

# At this point we have two representations of our data an "ff" (data.x), and
# a filehash representation (db).  Both can be indexed in similar fashions.
# However, the "ff" representation may be less familiar and some stock functions
# may not work as expected.  The "filehash" representation, is for the most part
# invisible to the user.

# > ls()
# [1] "data.x"   "db"       "i"        "lm.fit.1" "lsos"

lsos()
memory.size()
gc()
lsos()
memory.size()

# > lsos()
# Type               Size     Rows        Columns
# data.x         ffdf 131,784 1012027      62
# lm.fit.1      bigglm  19,956      13      NA
# lsos        function   2,900      NA      NA
# db       filehashDB1   1,900      62      NA
# i            integer      32       1      NA
#
# > memory.size()
# [1] 69.9
#
# > gc()
# used (Mb) gc trigger (Mb) max used  (Mb)
# Ncells  813586 21.8    2380321 63.6  2959476  79.1
# Vcells 1706952 13.1    6110221 46.7 20574192 157.0
```

```
#
# > lsos()
# Type                Size    Rows         Columns
# data.x          ffdf 131,784 1012027      62
# lm.fit.1      bigglm  19,956      13      NA
# lsos        function   2,900      NA      NA
# db       filehashDB1   1,900      62      NA
# i            integer      32       1      NA
#
# > memory.size()
# [1] 46.72
#

# List variable names in db
names(db)

# > names(db)
# [1] "sys.bp"               "bmi"                  "q6.physical.2"
# [4] "q3.cognitive.3"       "q13.social.4"         "glucose"
# [7] "animal.extinction"    "religion"             "social.responsibility"
# [10] "nat.gas.car"         "solar"                "q12.social.3"
# [13] "walk"                "coal"                 "q4.cognitive.4"
# [16] "wind"                "ice.melt"             "healthcare"
# [19] "gasoline"            "abortion"             "business.regulation"
# [22] "q11.social.2"        "neuroticism"          "q9.physical.5"
# [25] "region"              "q10.social.1"         "gender"
# [28] "tobacco.user"        "q7.physical.3"        "openness"
# [31] "agreeableness"       "bus"                  "national.debt"
# [34] "extroversion"        "q5.physical.1"        "other"
# [37] "severe.storms"       "ldl.cholesterol"      "age"
# [40] "sea.rise"            "bicycle"              "q1.cognitive.1"
# [43] "nuclear"             "id"                   "campaign.finance"
# [46] "public.edu"          "blood.type"           "train"
# [49] "education"           "q2.cognitive.2"       "nat.gas.electric"
# [52] "city.names"          "conscientiousness"    "electric"
# [55] "wbc.count"           "q8.physical.4"        "unemployment"
# [58] "hybrid"              "automobile"           "q14.social.5"
# [61] "income"              "plant.extinction"


table(db$gender)

# > table(db$gender)
# female   male
# 514927 497100

plot(density(db$income))
mean(db$ice.melt)
plot(density(db$ice.melt))

lsos()
memory.size()
gc()
lsos()
```

```
memory.size()

# > lsos()
# Type                Size    Rows        Columns
# data.x          ffdf 131,784 1012027      62
# lm.fit.1       bigglm  19,956      13      NA
# lsos        function   2,900      NA      NA
# db       filehashDB1   1,900      62      NA
# i            integer      32       1      NA
#
# > memory.size()
# [1] 74.21
# > gc()
# used (Mb) gc trigger (Mb) max used  (Mb)
# Ncells  821128 22.0    2380321 63.6  2959476  79.1
# Vcells 1716487 13.1    6246918 47.7 20574192 157.0
#
# > lsos()
# Type                Size    Rows        Columns
# data.x          ffdf 131,784 1012027      62
# lm.fit.1       bigglm  19,956      13      NA
# lsos        function   2,900      NA      NA
# db       filehashDB1   1,900      62      NA
# i            integer      32       1      NA
#
# > memory.size()
# [1] 46.77


# Here we fit a linear model using function "lm".  Notice
# that the size of the resulting output (when assigned to the R workspace)
# is quite large.  We will want to create a "filehash" database for the
# output as well, to keep the overall RAM assignment to R as low as possible.
# The nice thing about package" filehash" is that "list" type data structures
# can be assigned to a database as well.  This is important since many
# function's output are of a "list" data type.

system.time(lm.fit.2<-lm(db$income ~ db$gender))
summary(lm.fit.2)

# > system.time(lm.fit.2<-lm(db$income ~ db$gender))
# user  system elapsed
# 12.54    0.06   13.25

lsos()
memory.size()
gc()
lsos()
memory.size()


# > lsos()
# Type                Size        Rows        Columns
# lm.fit.2          lm 161,934,768      13      NA
```

```
# data.x          ffdf     131,784 1012027      62
# lm.fit.1      bigglm      19,956      13      NA
# lsos        function       2,900      NA      NA
# db       filehashDB1       1,900      62      NA
# i            integer          32       1      NA
#
# > memory.size()
# [1] 250.2
#
# > gc()
# used (Mb) gc trigger  (Mb) max used  (Mb)
# Ncells  1841039 49.2    2881618  77.0  2959476  79.1
# Vcells 10832037 82.7   27552109 210.3 27535040 210.1
#
# > lsos()
# Type               Size        Rows    Columns
# lm.fit.2          lm 161,934,768      13      NA
# data.x          ffdf     131,784 1012027      62
# lm.fit.1      bigglm      19,956      13      NA
# lsos        function       2,900      NA      NA
# db       filehashDB1       1,900      62      NA
# i            integer          32       1      NA
#
# > memory.size()
# [1] 185.14


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

# Creating a filehash database for function "lm" output


dbCreate("lmOutput.db")
db2<-dbInit("lmOutput.db")

system.time(db2$lm.fit.2<-lm(db$income ~ db$gender))

# > system.time(db2$lm.fit.2<-lm(db$income ~ db$gender))
# user  system elapsed
# 9.39    1.14   15.06

hist(db2$lm.fit.2$residuals)
mean(db2$lm.fit.2$residuals)


# Notice that the output list can be manipulated using
# conventional indexing operators, event though it is filehash
# database residing on the physical disk.

# Extracting the residuals of the linear model fit and calculating the
# mean of the residuals:

# > mean(db2$lm.fit.2$residuals)
# [1] -2.469686e-12
```

```
lsos()
memory.size()
gc()
lsos()
memory.size()
rm(lm.fit.2)
lsos()
gc()
memory.size()

# After garbage collection, we are back down to 47 MB, and we still
# have access to the output from the "lm" function if we need it - filehash
# object "db2".


# > lsos()
# Type               Size            Rows    Columns
# lm.fit.2         lm 161,934,768      13      NA
# data.x         ffdf     131,784 1012027      62
# lm.fit.1      bigglm      19,956      13      NA
# lsos        function       2,900      NA      NA
# db       filehashDB1       1,900      62      NA
# db2      filehashDB1       1,900       1      NA
# i            integer          32       1      NA
#
# > memory.size()
# [1] 223.28
#
# > gc()
# used (Mb) gc trigger  (Mb) max used  (Mb)
# Ncells  1842621 49.3    2881618  77.0  2959476  79.1
# Vcells 10831907 82.7   30353907 231.6 37870180 289.0
# > lsos()
# Type               Size            Rows    Columns
# lm.fit.2         lm 161,934,768      13      NA
# data.x         ffdf     131,784 1012027      62
# lm.fit.1      bigglm      19,956      13      NA
# lsos        function       2,900      NA      NA
# db       filehashDB1       1,900      62      NA
# db2      filehashDB1       1,900       1      NA
# i            integer          32       1      NA
# > memory.size()
# [1] 185.2
#
# > rm(lm.fit.2)
#
# > lsos()
# Type               Size    Rows        Columns
# data.x         ffdf 131,784 1012027      62
# lm.fit.1      bigglm  19,956      13      NA
# lsos        function   2,900      NA      NA
# db       filehashDB1   1,900      62      NA
# db2      filehashDB1   1,900       1      NA
```

```
# i          integer      32       1      NA
#
# > memory.size()
# [1] 185.56
#
# > lsos()
# Type               Size    Rows        Columns
# data.x         ffdf 131,784 1012027      62
# lm.fit.1      bigglm  19,956      13      NA
# lsos        function   2,900      NA      NA
# db       filehashDB1   1,900      62      NA
# db2      filehashDB1   1,900       1      NA
# i          integer      32       1      NA
#
# > gc()
# used (Mb) gc trigger  (Mb) max used  (Mb)
# Ncells  830528 22.2    2881618  77.0  2959476  79.1
# Vcells 1723689 13.2   24283125 185.3 37870180 289.0
#
# > memory.size()
# [1] 46.85

# Finally, we note what is necessary to re-populate our workspace with "data.x", "db",
# and "db2" without having to re-read the original text file, and re-initialize the "ff"
# and "filehash" databases.

# Reload the "filehash" and "ff" external files after closing R and reopening R:

require(ff)
require(ffbase)
require(filehash)

# For package "ff", the function "ffload" loads the internal "ff" object - "data.x".
# For package "filehas", the assignment db<-dbInit("Examplonia.db") relassigns the
# internal representation of the filehash database on physical disk.  Reassigning the
# previous output would be optional if desired.

source("ls.objects.r")
ffload("Examplonia.ff")
db<-dbInit("Examplonia.db")
db2<-dbInit("lmOutput.db")
lsos()


# On restarting R and assinging the "ff" and "filehash" objects, we have
# used approximately 30 M of RAM.

# > lsos()
#
# Type    Size    Rows Columns
# data.x       ffdf 131,784 1012027      62
# lsos     function   2,900      NA      NA
# db    filehashDB1   1,900      62      NA
# db2   filehashDB1   1,900       1      NA
```

```
#
# > memory.size()
# [1] 30.49
```