Big data in R

Nazia and Fentaw

University of Groningen
JBI of Mathematics and Computer Science

April 15, 2013

Outline

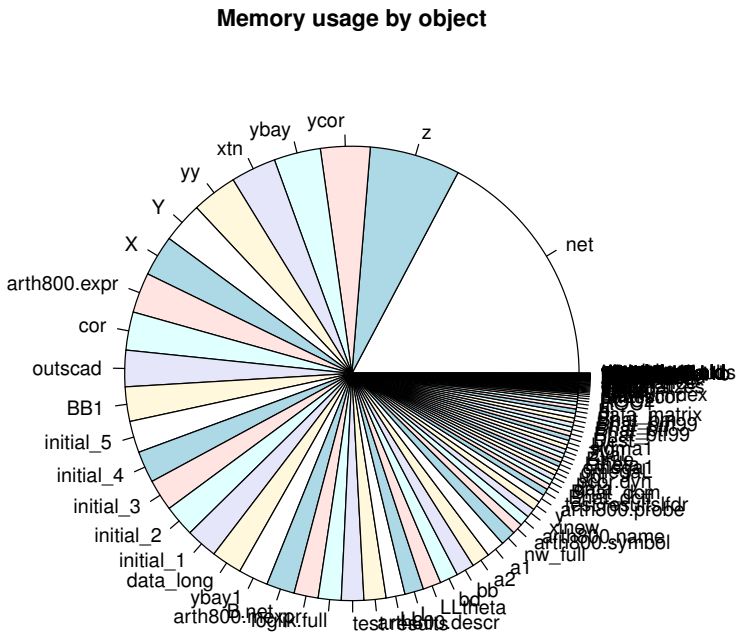
- R holds all objects in memory
- Limited amount of memory that can be used by all objects
- Memory error message while working with big data in R
- "cannot allocate vector of size _ MB"

```
memory.size()     % amount currently in use
memory.limit()    % memory limit
[1] 1535  MB
memory.limit(size=1800) % increase memory limit  (?)
[1] 1800 MB
```

- R objects and memory usage (my experience)

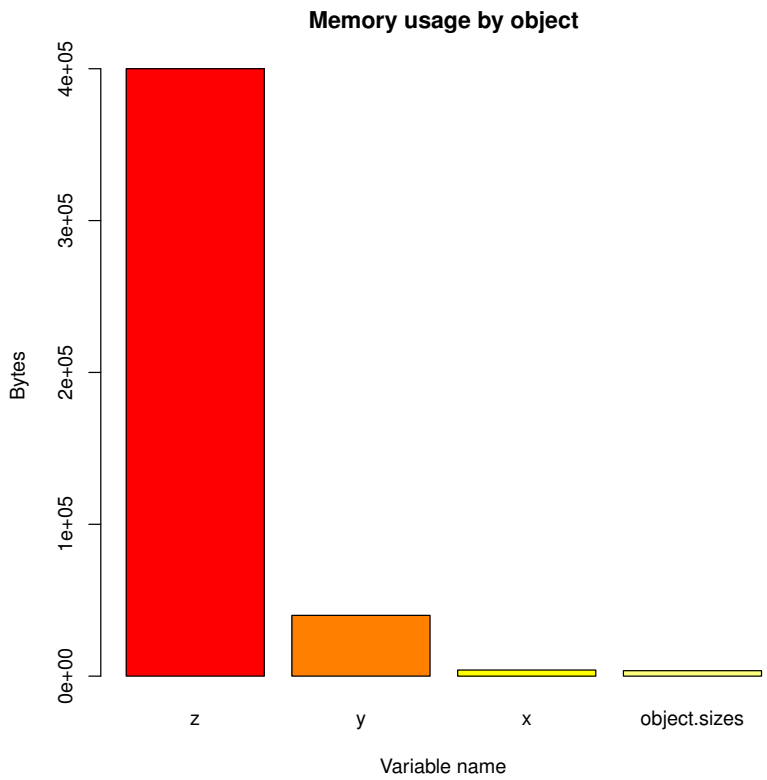
```
object.sizes()
       net              z           ycor           ybay
   1071880         400024         218688         205824
       xtn             yy              Y              X
    200120         200112         180120         180120
arth800.expr          cor        outscad            BB1
    177656         168580         160552         152192
  initial_5      initial_4      initial_3      initial_2
    139356         139356         139356         139356
  initial_1      data_long          ybay1          B.net
    139356         135168         131488         124040
arth800.mexpr  loglik.full  test.results  arth800.descr
    106824         101308          99008          97240
        LL              L       LLtheta             bd
     83556          83244          80200          80112
        bb             a2             a1        nw_full
     80112          80112          80112          67360  ...
```

Memory usage and R



```
rm(list=ls())
# create dummy variables for memory usage demonstration
x <- 1:1000
y <- 1:10000
z <- 1:100000
object.sizes <- function()
{
  return(rev(sort(sapply(ls(envir=.GlobalEnv),
      function (object.name)
      object.size(get(object.name))))))
}
object.sizes()
         z              y              x object.sizes
    400024          40024           4024          3580
```

Memory usage and R



- Data types stored as R object, eg. matrix

```
z <- matrix(0,1e+06,3)  ##numeric/double type

object.size(Z)
24000112 bytes  #24MB


W <- matrix(as.integer(0),1e+06,3) ##integer
object.size(w)
12000112 bytes #12MB
```

Big data packages

- biglm
- bigmemory, biganalytics, bigtabulate
- snow
- ff
- filehash
- mapReduce
- HadoopStreaming, etc.

biglm package: bounded memory linear and generalized linear models

Question: How it works?

- load data into memory in chunks
- process loaded data and update the sufficient statistic required for the model
- dispose the loaded chunk and load the next chunk
- repeat until end of file

```
bigglm(formula, data, family, ...,chunksize=5000)
```

- formula A model formula
- data eg. data frame
- family A glm family object: gaussian, gamma, poisson
- chunksize Size of chunks for processng the data frame

⋮

```
mydat_r <- matrix(rnorm(15000000,0,2),5000000,3)
colnames(mydat_r) <- c("first","second","third")
mydat_rdf <- as.data.frame(mydat_r)
> object.size(mydat_rdf)
   120MB
##  fit linear model using lm

linear.mod <- lm( first ~ second + third, data=mydat_rdf)
    Error: cannot allocate vector of size 114.4 Mb

## fit linear model using biglm

biglinear.mod <- bigglm( first ~ second + third,
     data=mydat_rdf, family=gaussian(), chunksize=100000)
```

```
> summary(biglinear.mod)

  Large data regression model: bigglm(first ~ second + third,
         data = mydat_rdf, family = gaussian(),
         chunksize = 1e+05)
      Sample size =  5e+06
                Coef    (95%    CI)    SE      p
(Intercept) -2e-04 -0.0020 0.0016 9e-04 0.8198
second       3e-04 -0.0006 0.0012 4e-04 0.4925
third       -6e-04 -0.0014 0.0003 4e-04 0.2165

    elapsed time is 21.370000 seconds
```

bigmemory package: manage massive matrices with shared memory and memory-mapped files

- bigmemory package allows powerful and memory-efficient analyses
- Permits storage of large objects (matrices, etc) in memory (on the RAM) using pointer objects to refer them
- Data sets may be file-backed (stored on har drive/disk), to easily manage and analyze datasets larger than available RAM
- several R processes on the same computer can also share big memory objects

bigmemory package

- bigmemory creates a variable $X <-$ big.matrix,
- X is apointer to the dataset saved in the RAM or the hard drive
- when X is passed as an argument to a function, it is essentially providing `call-by reference` rather than `call-by-value` behavior
- big.matrix is an R object that simply points to a data structure in C++
- The bigmemory package allows users to create matrices that are stored on disk, rather than in RAM. When an element is needed, it is read from the disk and cached in RAM.

```
x <- big.matrix(nrow, ncol,
    type = options()$bigmemory.default.type,
    init = NULL, dimnames = NULL, separated = FALSE,
    backingfile = NULL, backingpath = NULL,
    descriptorfile = NULL,
    shared = TRUE)

x <- filebacked.big.matrix(nrow, ncol,
    type = options()$bigmemory.default.type, init = NULL,
    dimnames = NULL, separated = FALSE,
    backingfile = NULL,
    backingpath = NULL, descriptorfile = NULL)

x <- as.big.matrix(x, type = NULL, separated = FALSE,
     backingfile = NULL, backingpath = NULL,
     descriptorfile = NULL,
     shared=TRUE)
```

- big.matrix is local to a single R process and is limited by available RAM
- big.matrix(shared.true) is like big.matrix but can be shared among multiple R processes.
- filebacked.bigmatrix : point to a file on disk containing the matrix and the file can be shared across clusters.

bigmemory package

- x a matrix, vector, or data.frame for as.big.matrix;
- nrow number of rows.
- ncol number of columns.
- type the type of the atomic element (options: “double", “integer", “short", or“char").
- init a scalar value for initializing the matrix

bigmemory package

- dimnames a list of the row and column names; use with caution for large objects.
- separated use separated column organization of the data.
- backingfile the root name for the file(s) for the cache of x.
- backingpath the path to the directory containing the file backing cache.
- descriptorfile the name of the file to hold the backingfile description, for subsequent use with attach.big.matrix.
- shared if TRUE, the resulting big.matrix can be shared across processes.

```
    > z <- big.matrix(3, 4, type="integer", init=5)
    > z
    An object of class "big.matrix"
    Slot "address":
     <pointer: 0x06148810>
> z[,]
      [,1] [,2] [,3] [,4]
[1,]    5    5    5    5
[2,]    5    5    5    5
[3,]    5    5    5    5
 > dim(z)
[1] 3 4
 > z[2,2] <- as.integer(3)
> z[,]
      [,1] [,2] [,3] [,4]
[1,]    5    5    5    5
[2,]    5    3    5    5
[3,]    5    5    5    5
```

```
 > zz <- filebacked.big.matrix(3, 4, type="integer", init=5,
+ backingfile="example0.bin", descriptorfile="example0.desc",
+ dimnames=NULL)    ##shared =TRUE always
> zz
An object of class "big.matrix"
Slot "address":
<pointer: 0x017d4238>

> zz[,]
     [,1] [,2] [,3] [,4]
[1,]    5    5    5    5
[2,]    5    5    5    5
[3,]    5    5    5    5
```

Open a second R session

```
 > yy <- attach.big.matrix("example0.desc")
> yy
An object of class "big.matrix"
Slot "address":
<pointer: 0x017d46d8>

> yy[,]
     [,1] [,2] [,3] [,4]
[1,]    5    5    5    5
[2,]    5    5    5    5
[3,]    5    5    5    5
```

Second R session

```
 > yy[2,2] <- 100   ##change a value

> yy[,]
     [,1] [,2] [,3] [,4]
[1,]    5    5    5    5
[2,]    5  100    5    5
[3,]    5    5    5    5

 > Check the first R session
 > Notice that zz = yy
```

```
write.big.matrix(x, filename, row.names = FALSE,
      col.names = FALSE, sep=‘‘,’’)

read.big.matrix(filename, sep = ‘‘,’’ , header = FALSE,
      col.names = NULL, row.names = NULL,
      has.row.names=FALSE, ignore.row.names=FALSE,
      type = NA, skip = 0, separated = FALSE,
      backingfile = NULL, backingpath = NULL,
      descriptorfile = NULL, extraCols = NULL,
      shared=TRUE)
```

```
     # Without specifying the type, big.matrix x will hold integers.
> x <- as.big.matrix(matrix(1:10, 5, 2))
> x[2,2] <- NA
> x[,]
     [,1] [,2]
[1,]    1    6
[2,]    2   NA
[3,]    3    8
[4,]    4    9
[5,]    5   10
> write.big.matrix(x, "sample.txt")

 #Read it back in as character (1-byte integers):
> y <- read.big.matrix("sample.txt", type="char")
> y[,]
     [,1] [,2]
[1,]    1    6
[2,]    2   NA
[3,]    3    8
[4,]    4    9
[5,]    5   10
```

Pitfalls of big.matrix

- only one type of data structure
- nonnumeric data values (no guarantee)

```
 > w <- as.big.matrix(matrix(1:10, 5, 2), type="double")
> w[,]
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
> w[1,2] <- NA
> w[2,2] <- -Inf
> w[3,2] <- "b"
Warning message:
In SetElements.bm(x, i, j, value) : NAs introduced by coercion
> w[4,2] <- NaN
> w[,]
     [,1] [,2]
[1,]    1   NA
[2,]    2 -Inf
[3,]    3   NA
[4,]    4  NaN
[5,]    5   10
```

```
> write.big.matrix(w, "nonnumeric.txt")
> w1 <- read.big.matrix("nonnumeric.txt", type="double")
> w1[,]
     [,1] [,2]
[1,]    1   NA
[2,]    2   -1
[3,]    3   NA
[4,]    4   NA
[5,]    5   10
> w2 <- read.big.matrix("nonnumeric.txt", type="integer")
> w3 <- read.big.matrix("nonnumeric.txt", type="char")
> w4 <- read.big.matrix("nonnumeric.txt", type="short")
```

```
  library(bigmemory)
  library(biganalytics)
  mydata.big = as.big.matrix(mydat_r) ## from R matrix
> mydata.big
An object of class "big.matrix"
Slot "address":
<pointer: 0x06109940>
 > options(bigmemory.allow.dimnames=TRUE)
        ##permission to change col names
 > colnames(mydata.big) = c("first","second","third")
 > head(mydata.big)
          first     second      third
[1,] -1.8358754 -2.9763708  0.3542316
[2,] -1.4135088  1.5770370 -1.1113071
[3,] -2.0222139 -1.4661885 -4.0510420
[4,] -2.3542718  1.1143254  5.1107734
[5,]  0.9703098  3.4126123  0.5251787
[6,]  0.3535078 -0.7306655  1.3299772
```

```
 ##  fit linear model using biglm.big.matrix

>bigmatrix.linear.mod <- biglm.big.matrix(first ~ second + third
           data=mydata.big)
>  summary(bigmatrix.linear.mod)
   Large data regression model: biglm(formula = formula,
            data = data, ...)
        Sample size =  5000000
              Coef    (95%    CI)    SE      p
(Intercept) -2e-04 -0.0020 0.0016 9e-04 0.8198
second       3e-04 -0.0006 0.0012 4e-04 0.4925
third       -6e-04 -0.0014 0.0003 4e-04 0.2165

elapsed time is 7.300000 seconds
```

Thank you!