Deal with big data in R using bigmemory package

Xiaojuan Hao

Department of Statistics
University of Nebraska-Lincoln

April 28, 2015

Background

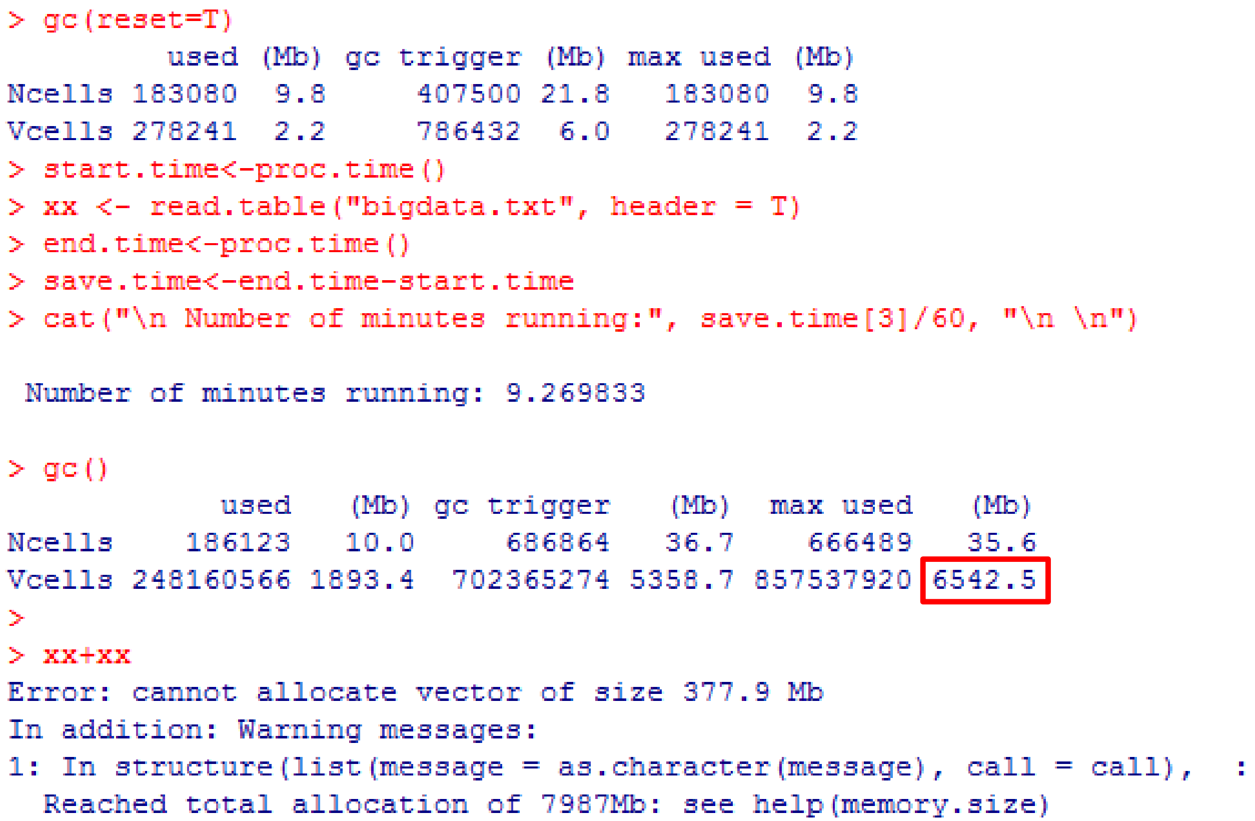
What Is Big Data

- Size (We focus on)
- Complexity
- Rate of growth

Problem In R For The Big Size Of Data

- Size of 2 GB
- Runs on a windows 7 system with 8 GB of RAM.
- Takes about 9 minutes with `read.table` function
- Consume 6 GB of memory.
- Some simple calculations will cause error:

  Error message : Reached total allocation of 7987Mb: see help(memory.size)



```
> gc(reset=T)
         used (Mb) gc trigger (Mb) max used (Mb)
Ncells 183080  9.8     407500 21.8   183080  9.8
Vcells 278241  2.2     786432  6.0   278241  2.2
> start.time<-proc.time()
> xx <- read.table("bigdata.txt", header = T)
> end.time<-proc.time()
> save.time<-end.time-start.time
> cat("\n Number of minutes running:", save.time[3]/60, "\n \n")

 Number of minutes running: 9.269833

> gc()
             used   (Mb) gc trigger   (Mb)  max used   (Mb)
Ncells     186123   10.0     686864   36.7    666489   35.6
Vcells 248160566 1893.4  702365274 5358.7 857537920 6542.5
>
> xx+xx
Error: cannot allocate vector of size 377.9 Mb
In addition: Warning messages:
1: In structure(list(message = as.character(message), call = call),  :
  Reached total allocation of 7987Mb: see help(memory.size)
```

1. You can't load the data in R with only 4 GB of memory.
2. The `gc` function is used to monitor the usage of memory .
3. The element in the last column and second rows shows the maximum space used in Mb since the last call to `gc(reset = TRUE)`

❖ About R

❑ We love R because

- Flexible for data analysis
- Big graphical capabilities
- Extensible
- Free and available for different platforms.

❑ Two major limitations

- Uses only 1 core by default on CPU
- R reads data into memory by default

1. Exhaust RAM by storing unnecessary data.
2. More memory will lead system to be frozen.

❖Couple of Solutions

- Buy more RAM

  Expensive

- Apply more memory on HCC

  Not convenient

- Use C/C++ program

  Lack the flexibility and convenience of R’s rich environment.

- Use R packages for big data support

  1. bigmemory 2. ff

- Use parallelism to process and generate large data sets on cluster

  1. Hadoop 2. MapReduce

➢ Outline

❖ Introduce bigmemory Package

- Why we need bigmemory
- About the big family

❖ How To Use bigmemory Package

- Some main functions.
- Shared memory with foreach iteration
- Use biglm package with bigmemory
- Other commands

❖ Summary of The bigmemory Package

➢ Introduce bigmemory

❖ What Is bigmemory

- Manage massive matrices with shared memory and memory-mapped file

❖ Why We Need bigmemory

- Store big matrices in memory and support their basic manipulation and exploration. This procession was managed in R but implemented in C++
- Very simple to use
- Multiple processors on the same machine can access to the same big data sets by shared memory.

❖ Big Family

The big family contains several packages for analysis of big datasets

- **bigmemory**

  Provides the core matrix-like support.

  Functions include: `nrow,ncol,dim,tail,head,apply,big.matrix,`

  `read.big.matrix, mwhich.`

- **biganalytics**

  Provides routine analysis on big matrix.

  Functions include: s`um,range,mean,colsum,colrange,colmean,`

  `biglm.big.matrix,bigglm.big.matrix`

- **bigtabulate**

  Adds table and split-like support for `big.matrix` objects

- **bigalgebra**

  Provides linear algebra operations on R `matrix` as well as `big.matrix`.

➢ Some Main Functions

❖ Read In Data Functions

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

type: integer (4 bytes), short(2 bytes), double(8 bytes), char(1 bytes)

backingfile: the root name for the file(s) for the cache of x.

backingpath: the path to the directory containing the backingfile

descriptorfile: the name of the file to hold the backingfile description.

shared: if TRUE, the `big.matrix` can be shared across processes.

Example:

Step 1: Simulate one data set

```
mydata=matrix(c(NA),nrow=10072112,ncol=5)
set.seed(12345)
mydata[,1]=sample(c(1:17770), 10072112, replace = TRUE)
mydata[,2]=sample(c(1:480189), 10072112, replace = TRUE)
mydata[,3]=sample(c(1:5), 10072112, replace = TRUE)
mydata[,4]=sample(c(1999:2005), 10072112, replace = TRUE)
mydata[,5]=sample(c(1:12), 10072112, replace = TRUE)
write.table(mydata, file = "example.txt", sep = " ",row.names
= F, col.names = F)
```

- 10 million rows and 5 columns
- File has size of 215.1055 MB

Step 2: Read in the “example.txt” file

1. Use usual R `read.table` function

```
> gc(reset=T)
         used  (Mb)  gc trigger  (Mb)  max used  (Mb)
Ncells 183080   9.8      407500 21.8    183080   9.8
Vcells 278241   2.2      786432  6.0    278241   2.2
> start.time<-proc.time()
> x <- read.table("example.txt",colClasses = "integer", header=F,
        col.names = c("movie", "customer", "rating","year",
                      "month"))
> end.time<-proc.time()
> save.time<-end.time-start.time
> cat("\n Number of minutes running:", save.time[3]/60, "\n \n")
 Number of minutes running: 0.6815
> gc()
  used   (Mb) gc trigger   (Mb) max used   (Mb)
N 186106  10.0     407500  21.8   188080  10.1
V 25463925 194.3   55646223 424.6 51920931 396.2
> dim(x)
[1] 10072112        5
```

2. Use `read.big.matrix` function

```
> gc(reset=T)
         used (Mb) gc trigger (Mb) max used (Mb)
Ncells 287878 15.4     467875   25   287878 15.4
Vcells 429260  3.3     905753    7   429260  3.3
> start.time<-proc.time()

➢ x <- read.big.matrix("example.txt", header =F,type = "integer",sep = "
         ",backingfile ="data.bin", descriptor = "data.desc",col.names =
          c("movie", "customer","rating","year", "month"), shared=TRUE)
> end.time<-proc.time()
> save.time<-end.time-start.time
> cat("\n Number of minutes running:",save.time[3]/60, "\n \n")
 Number of minutes running: 0.8

> gc()
         used (Mb) gc trigger (Mb) max used (Mb)
Ncells 290574 15.6     531268 28.4   303567 16.3
Vcells 432636  3.4     905753  7.0   905084  7.0
> dim(x)
[1] 10072112        5
```

a) Spend more time

b) Save memory

c) `read.big.matrix` function creates the binary file-backing associated with the `big.matrix` object x and shared descriptor file.

d) Use shared descriptor, we can load the data from disk with `attach.big.matrix` function.

```
> start.time<-proc.time()
> datadesc<-dget("data.desc")
> data<-attach.big.matrix(datadesc)
> end.time<-proc.time()
> save.time<-end.time-start.time
> cat("\n Number of minutes running:", save.time[3]/60, "\n \n")
   Number of minutes running: 0.0006666667
> data
An object of class "big.matrix"
Slot "address":
<pointer: 0x00000000001c9ec0>
> head(data)
     movie customer rating year month
[1,] 12811   121761      5 2004    12
[2,] 15563   351198      2 2005     9
[3,] 13523   173425      5 2002     3
[4,] 15747   132685      1 2003     4
[5,]  8112   401461      4 2003     9
[6,]  2957   346798      3 2000     3
```

e) `big.matrix` holds an pointer to a C++ matrix that is on disk

```
> x
An object of class "big.matrix"
Slot "address":
<pointer: 0x00000000003496b0>

> head(x)
     movie customer rating year month
[1,] 12811   121761      5 2004    12
[2,] 15563   351198      2 2005     9
[3,] 13523   173425      5 2002     3
[4,] 15747   132685      1 2003     4
[5,]  8112   401461      4 2003     9
[6,]  2957   346798      3 2000     3
> is.filebacked(x)
[1] TRUE
> is.big.matrix(x)
[1] TRUE
> is.shared(x)
[1] TRUE
```

```
> datadesc<-dget("data.desc")
> data<-attach.big.matrix(datadesc)
> data
An object of class "big.matrix"
Slot "address":
<pointer: 0x00000000001c9ec0>
> head(data)
     movie customer rating year month
[1,] 12811   121761      5 2004    12
[2,] 15563   351198      2 2005     9
[3,] 13523   173425      5 2002     3
[4,] 15747   132685      1 2003     4
[5,]  8112   401461      4 2003     9
[6,]  2957   346798      3 2000     3
> is.filebacked(x)
[1] TRUE
> is.big.matrix(x)
[1] TRUE
> is.shared(x)
[1] TRUE
```

f) The matrices can contain only numeric (char, short, int, double) values.

g) Need preprocess data so that factors are coded as numeric values.

Example: Read in test.csv file

i. Total 20 patients

ii. Three columns. (PID (integer), BP(factor), W (integer))

```
> test <- read.big.matrix("test.csv", header = T, type = "integer")
> test
An object of class "big.matrix"
Slot "address":
<pointer: 0x0000000019bcd100>

> head(test[,])
      PID BP   W
 [1,]   1 NA  90
 [2,]   2 NA 138
 [3,]   3 NA 170
 [4,]   4 NA 112
 [5,]   5 NA 130
 [6,]   6 NA 100
```

❖ Construct Matrix Functions

```
big.matrix(nrow, ncol, type, init = NULL, dimnames = NULL,
           separated = FALSE,backingfile = NULL,
           backingpath = NULL, descriptorfile = NULL,
           binarydescriptor=FALSE, shared = TRUE)
```

Eample 1

```
>a<- big.matrix(10, 2, type='integer', init=-5)
> options(bigmemory.allow.dimnames=TRUE)
> colnames(a) <- c("alpha", "beta")
> is.big.matrix(a)
[1] TRUE
> is.shared(a)
[1] TRUE
> is.filebacked(a)
[1] FALSE
```

Eample 2

```
> z <- filebacked.big.matrix(3, 3, type='integer', init=123,
       backingfile="example.bin", descriptorfile="example.desc",
       dimnames=list(c('a','b','c'), c('d', 'e', 'f')))
> is.big.matrix(z)
[1] TRUE
> is.shared(z)
[1] TRUE
> is.filebacked(z)
[1] TRUE
```

- **Several Matrix Objects**

1. `big.matrix`
   a) Points to a data structure in C++.
   b) For a single R process.
   c) Limited by available RAM

2. `shared.big.matrix`
   a) Similar to `Big.matrix`
   b) Can be shared among multiple R processes

3. `filebacked.big.matrix`
   a) Points to a file on disk containing the matrix
   b) Files can be shared across a cluster

❖ mwhich Function

- Provides efficient row selections for `big.matrix` and `matrix` objects.
- Based loosely on R's `which` function without memory overhead

```
> gc(reset=T)
         used (Mb) gc trigger (Mb) max used (Mb)
Ncells 292363 15.7     531268 28.4   292363 15.7
Vcells 436893  3.4     905753  7.0   436893  3.4
> cust.indices.inefficient <- which(data[, "customer"] == as.integer(6))
> gc()
         used (Mb) gc trigger (Mb) max used (Mb)
Ncells 293247 15.7     531268 28.4   303777 16.3
Vcells 437744  3.4    9539607 72.8 10669871 81.5
> head(data[cust.indices.inefficient,])
     movie customer rating year month
[1,]   508        6      1 2004     4
[2,]  2652        6      2 1999     8
[3,]  1171        6      2 2001    10
[4,] 15662        6      2 2001     7
[5,]  7139        6      5 2001     7
[6,]   858        6      3 2000     5
```

`mwhich(x, cols, vals, comps, op = 'AND')`

x: a `big.matrix` object.

cols: a vector of column indices or names.

vals: a list of vectors of length 1 or 2; length 1.It is used to test equality (or inequality),

comps: a list of operators including 'eq', 'neq', 'le', 'lt', 'ge' and 'gt‘.

op: either 'AND' or 'OR'.

```
> gc(reset=T)
         used (Mb) gc trigger (Mb) max used (Mb)
Ncells 293350 15.7     531268 28.4   293350 15.7
Vcells 437916  3.4    7631685 58.3   437916  3.4

> cust.indices <- mwhich(data, "customer", 6, "eq")

> gc()
         used (Mb) gc trigger (Mb) max used (Mb)
Ncells 294316 15.8     531268 28.4   301658 16.2
Vcells 438252  3.4    6105348 46.6   771162  5.9
```

```
> sum(cust.indices.inefficient != cust.indices)
[1] 0
> head(data[cust.indices, ])
     movie customer rating year month
[1,]   508        6      1 2004     4
[2,]  2652        6      2 1999     8
[3,]  1171        6      2 2001    10
[4,] 15662        6      2 2001     7
[5,]  7139        6      5 2001     7
[6,]   858        6      3 2000     5

> these <- mwhich(data, c("customer", "rating"), list(6, 2),
                  list("eq","le"), "AND")
> head(data[these, ])
     movie customer rating year month
[1,]   508        6      1 2004     4
[2,]  2652        6      2 1999     8
[3,]  1171        6      2 2001    10
[4,] 15662        6      2 2001     7
[5,]  4035        6      2 2002     4
[6,]  1970        6      1 1999    10
```

❖ deepcopy Function

```
deepcopy(x, cols = NULL, rows = NULL, y = NULL,
            type = NULL,separated = NULL, backingfile =
            NULL,backingpath = NULL, descriptorfile =
            NULL,binarydescriptor=FALSE, shared=TRUE)
```

```
> z
An object of class "big.matrix"
Slot "address":
<pointer: 0x00000000003c9820>

> z[,]
    d   e   f
a 123 123 123
b 123 123 123
c 123 123 123
> y<-z
> y
An object of class "big.matrix"
Slot "address":
<pointer: 0x00000000003c9820>
```

Same

```
> y[,]
    d   e   f
a 123 123 123
b 123 123 123
c 123 123 123
> y[1,1]<-as.integer(1)
> z[,]
    d   e   f
a   1 123 123
b 123 123 123
c 123 123 123
> w<-deepcopy(z)
> w
An object of class "big.matrix"
Slot "address":
<pointer: 0x00000000002cd2a0>
```

z and y point to the same data in memory.

❖ flush Function

- `flush()` forces any modified information to be written to the file-backing.

```
> library(bigmemory)
> w <- read.big.matrix("w.txt", header
      = F, type = "integer",sep = " ",
       backingfile = "w.bin",
       descriptor = "w.desc",
       col.names = c("a", "b", "c"))
> w[,]
       a   b   c
[1,] 123 123 123
[2,] 123 123 123
> w[,1]<-1
> w[,]
     a   b   c
[1,] 1 123 123
[2,] 1 123 123
> flush(w)
> neww=attach.big.matrix(dget("w.desc"))
> neww[,]
     a   b   c
[1,] 1 123 123
[2,] 1 123 123
```

The backed file is changed

```
> w=read.big.matrix("w.txt", header = F, type = "integer",sep = " ")
> w[,]
     [,1] [,2] [,3]
[1,]  123  123  123
[2,]  123  123  123

> neww=attach.big.matrix(dget("w.desc"))
> neww[,]
     a   b   c
[1,] 1 123 123
[2,] 1 123 123
```

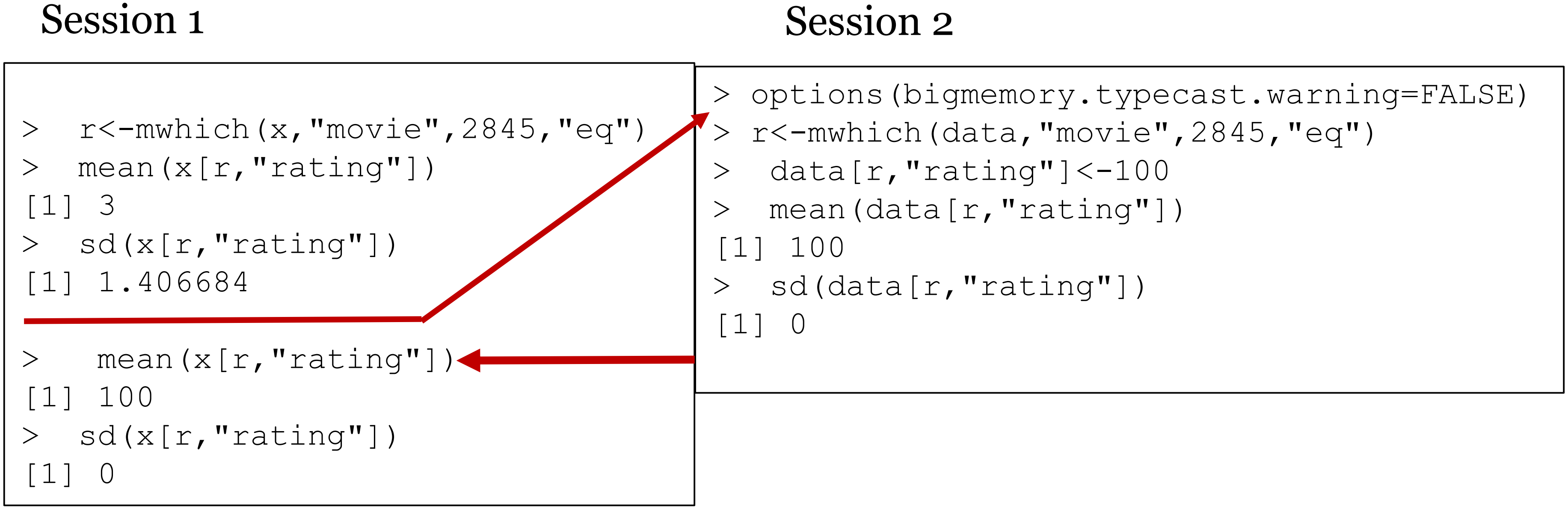
- It only changes the backed file.
- The original data set does not be changed
- The backed file does not match the original data anymore !!!

Shared Memory

Interactive Shared Memory

Example 1

1. Two R sessions are connected to the same `shared.big.matrix`
2. The assignment in one process will affect the value in the other sessions.



Session 1

```
>  r<-mwhich(x,"movie",2845,"eq")
>  mean(x[r,"rating"])
[1] 3
>  sd(x[r,"rating"])
[1] 1.406684

>   mean(x[r,"rating"])
[1] 100
>  sd(x[r,"rating"])
[1] 0
```

Session 2

```
> options(bigmemory.typecast.warning=FALSE)
> r<-mwhich(data,"movie",2845,"eq")
>  data[r,"rating"]<-100
>  mean(data[r,"rating"])
[1] 100
>  sd(data[r,"rating"])
[1] 0
```

❖ Parallel Computation With bigmemory

Example 2

1. Calculate the average rate for each movie.
2. Takes about 2.5 hours using for loop iteration.

```
> start.time<-proc.time()
> movie_uniq<-data[unique(data[,1]),1]
> n<-length(movie_uniq)
> movie_av_rate<-big.matrix(n,2, type='double')
> movie_av_rate[,1]<-movie_uniq
> for (i in 1:n)
  movie_av_rate[i,2] <-
     mean(data[mwhich(data,"movie",movie_uniq[i],"eq"),"rating"])
> end.time<-proc.time()
> save.time<-end.time-start.time
> cat("\n Number of minutes running:", save.time[3]/60, "\n \n")

 Number of minutes running: 137.1643
```

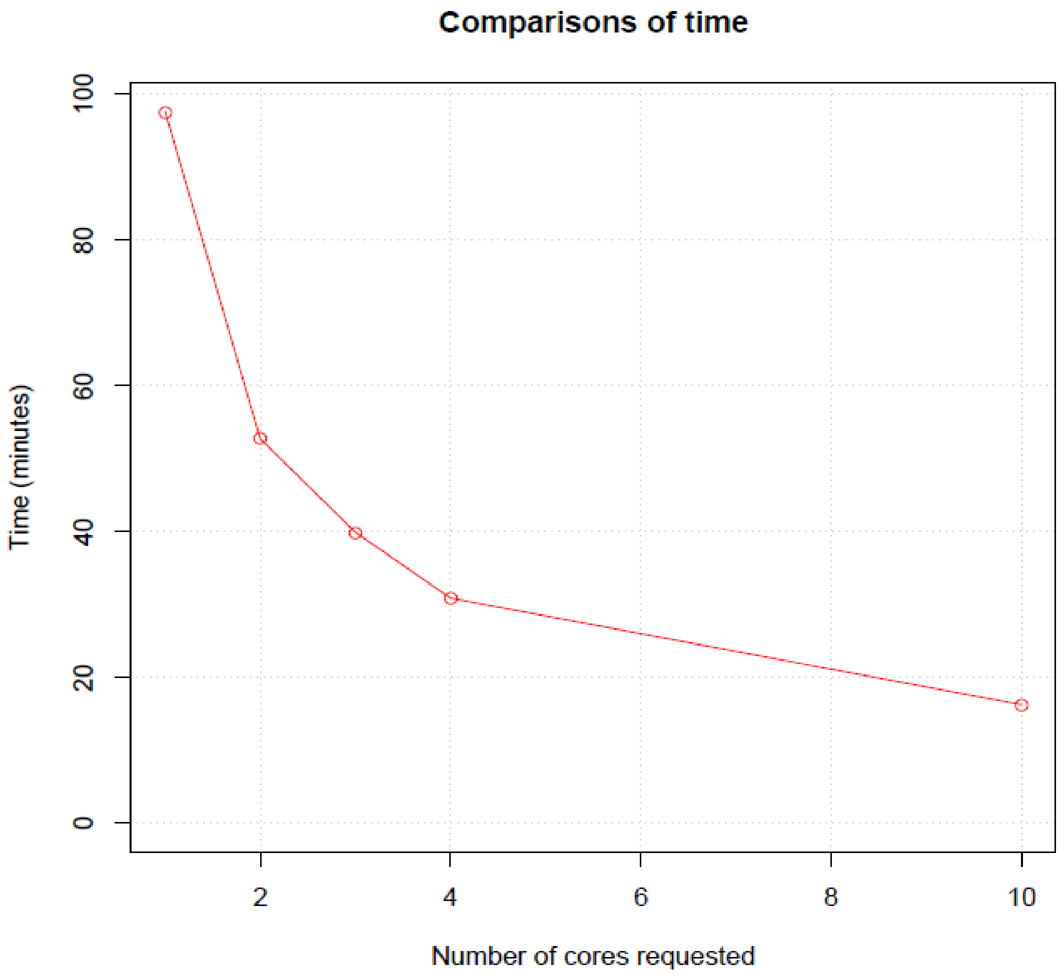
3. Takes about 1.5 hours for parallel computation with two cores

```
> library(bigmemory)
> datadesc<-dget("data.desc")
> data<-attach.big.matrix(datadesc)
> movie_uniq<-data[unique(data[,1]),1]
> n<-length(movie_uniq)
> movie_av_rate<-big.matrix(n,2, type='double')
> movie_av_rate[,1]<-movie_uniq
>
> library(doParallel)
> cl<-makeCluster(spec = 2)
> registerDoParallel(cl = cl)
> library(foreach)
> start.time<-proc.time()
> clusterSetRNGStream(cl = cl, iseed = 9182)
> res<-foreach(i = 1:n,.combine = rbind) %dopar% {
      require(bigmemory)
      data<-attach.big.matrix(datadesc)
      mean(data[mwhich(data,"movie",movie_uniq[i],"eq"),"rating"])}
> stopCluster(cl)
> movie_av_rate[,2]<-res
> end.time<-proc.time()
> save.time<-end.time-start.time
> cat("\n Number of minutes running:", save.time[3]/60, "\n\n")
  Number of minutes running: 81.845
```

Takes about 1.5 hours

4. Use Crane

Time in minutes vs. the number of threads



➢ biglm Package With bigmemory

- `biganalytics` provides linear and generalized linear models for big data based on `biglm` package.

- `biglm.big.matrix` and `bigglm.big.matrix` save memory than usual `lm` function for big data regression.

Example

For movie data, we try to predict the customer ratings using the factor of movie released year

```
library(biglm)
# library(biganalytics)
lm.b<-biglm.big.matrix(rating ~ year, data=x, fc="year")
```

```
> summary(lm.b)
Large data regression model: biglm(formula = formula, data = data, ...)
Sample size =  10072112
                Coef    (95%    CI)      SE       p
(Intercept)   3.0072  3.0045 3.0098 0.0013 0.0000
year2000     -0.0024 -0.0061 0.0014 0.0019 0.2069
year2001     -0.0016 -0.0054 0.0021 0.0019 0.3824
year2002     -0.0027 -0.0065 0.0010 0.0019 0.1496
year2003      0.0001 -0.0036 0.0039 0.0019 0.9489
year2004     -0.0006 -0.0043 0.0032 0.0019 0.7643
year2005     -0.0009 -0.0046 0.0029 0.0019 0.6495
```

$$\widehat{Rating} = \hat{a} + \widehat{b_1} * year2000 + \widehat{b_2} * year2001 + \widehat{b_3} * year2002 + \widehat{b_4} * year2003$$

$$+ \widehat{b_5} * year2004 + \widehat{b_6} * year2005$$

- It appears that there is no linear relationship between the movie ratings and released years

Other Commands

- Write a `big.matrix` to a file using `write.big.matrix` function.
- Parallel `foreach` function with `bigmemeory` package will be the most efficient way to deal with big data sets.
- Save memory using correct type.

```
> gc(reset=T)
         used (Mb) gc trigger (Mb) max used (Mb)
Ncells 295907 15.9     531268 28.4   295907 15.9
Vcells 440279  3.4    4884278 37.3   440279  3.4
> data[,4]<-data[,4]+1
> gc()
           used (Mb) gc trigger  (Mb) max used  (Mb)
Ncells   296361 15.9     531268  28.4   301473  16.2
Vcells 10512493 80.3   30178098 230.3 35852849 273.6
> gc(reset=T)
         used (Mb) gc trigger  (Mb) max used (Mb)
Ncells 296369 15.9     531268  28.4   296369 15.9
Vcells 440408  3.4   24142478 184.2   440408  3.4
> data[,4]<-data[,4]-as.integer(1)
> gc()
          used (Mb) gc trigger  (Mb) max used  (Mb)
Ncells  296361 15.9     531268  28.4   299041  16.0
Vcells 5476437 41.8   19313982 147.4 20745705 158.3
```

➢ Summary of `bigmemory`

- Store a matrix in memory and easy to access without reload the data.
- Share matrices among multiple R sessions and clusters.
- Be careful when you use shared matrix.
- More efficient to use parallel calculation with bigmemory package for large data sets.
- Only deal with numeric matrices.
- More big family and functions are available.

See http://cran.r-project.org/web/packages/bigmemory/bigmemory.pdf

http://cran.r-project.org/web/packages/bigmemory/index.html

➢ Reference

- The Bigmemory Project, http://www.bigmemory.org/, the home of R packages bigmemory, biganalytics, bigtabulate, bigalgebra, and synchronicity. Packages available from CRAN or R-Forge.

- Emerson JW, Kane MJ (2009). “The R Package bigmemory: Supporting Efficient Computation and Concurrent Programming with Large Data Sets.” Journal of Statitical Software, Volume VV, Issue II.

- 2009 JSM Data Expo: Airline on-time performance. http://stat-computing.org/dataexpo/2009/.

- “The Bigmemory Project” by Michael Kane and John Emerson: April 29, 2010. http://cran.r-project.org/web/packages/bigmemory/vignettes/Overview.pdf

- “Taking R to the limit” by Ryan R Rosario: August 17,2010. http://www.bytemining.com/wp-content/uploads/2010/08/r_hpc_II.pdf