Your Best Next Business Solution

Big Data In R

24/3/2010

Big Data In R

- R Works on RAM
- Causing Scalability issues
- Maximum length of an object is 2^31-1
- Some packages developed to help overcome this problem

Big Data In R

- RODBC Package
- biglm Package
- ff Package
- bigmemory package
- snow package

RODBC Package

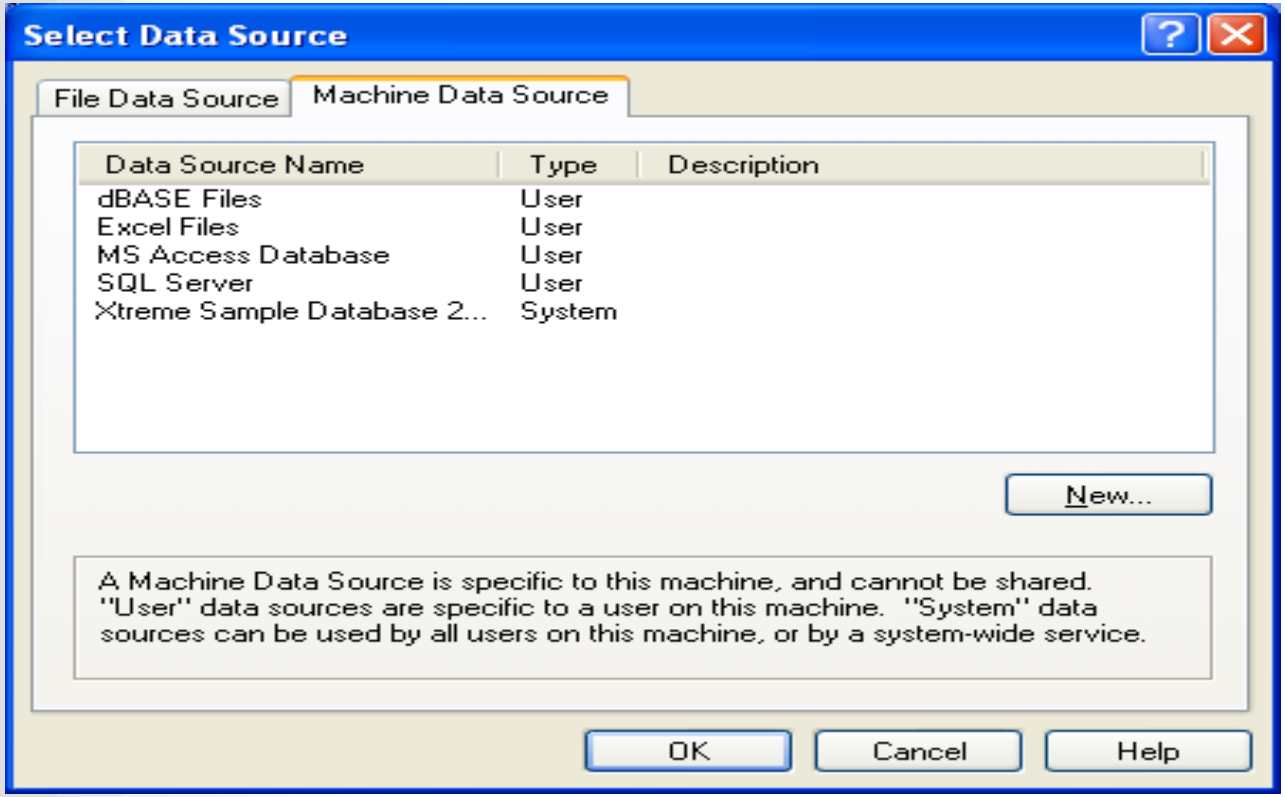
- Connecting to external DB from R to retrieve and handle data stored in the DB
- RODBC package support connection to SQL-based database (DBMS) such as: Oracle, SQL Server, SQLite, MySQL and more
- Require an ODBC driver which usually comes with the DBMS
- Windows offer an ODBC driver to flat files and Excel
- Supports client-server architecture

Source: http://cran.r-project.org/web/packages/RODBC/RODBC.pdf

RODBC Package

Defining DSN:

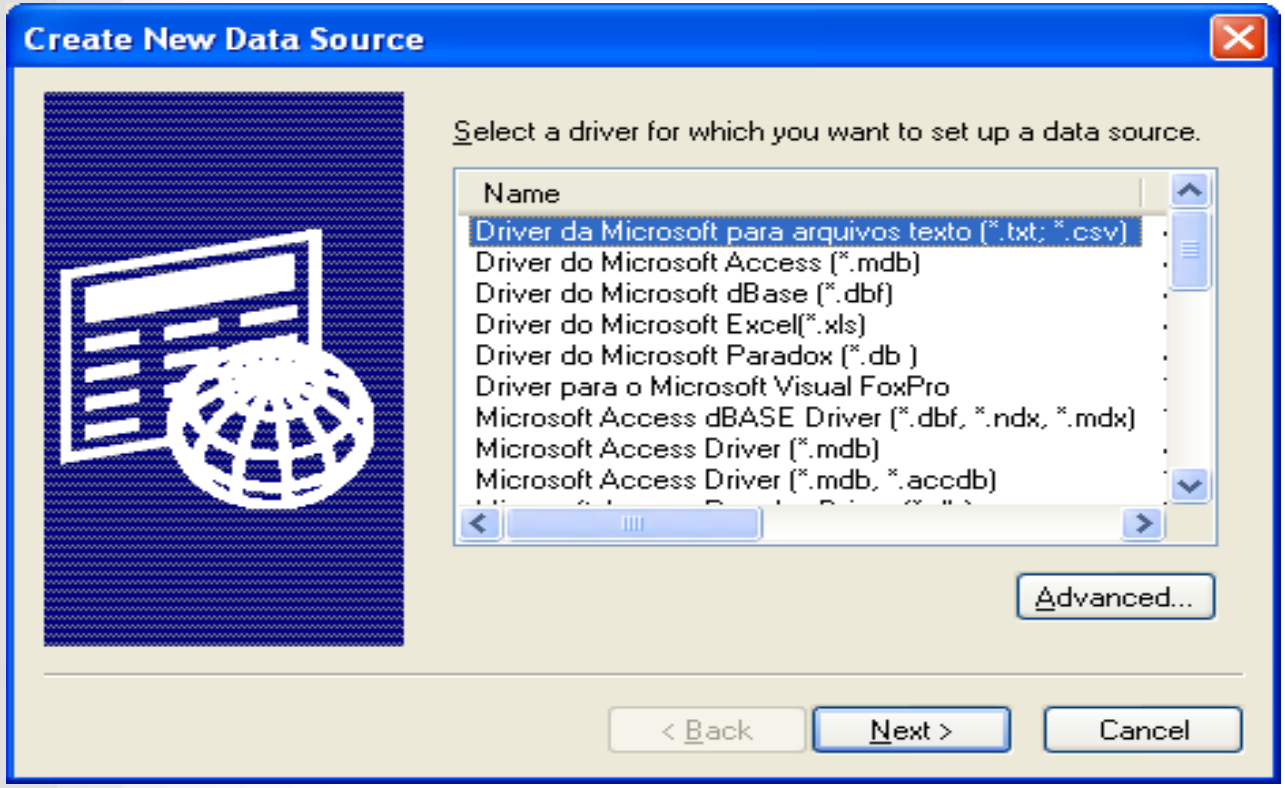
> odbcDriverConnect()



RODBC Package

Defining DSN:

> odbcDriverConnect()



RODBC Package

Main Commands:

odbcConnect(dsn, uid = "", pwd = "", ...)

odbcGetInfo(channel)

sqlColumns(channel, sqtable, ...)

sqlFetch(channel, sqtable, ..., colnames = FALSE, rownames = TRUE)

sqlQuery(channel, query, errors = TRUE, ..., rows_at_time)

RODBC Package

```
# Open connection:
    > xf <- odbcConnect(dsn="SQL Server")
# Read table:
    > go<-sqlFetch(xf,sqtable="adult“,colnames=T)
    > str(go)
    data.frame':   32560 obs. of  15 variables:….
# Alternatively:
    > go1 <- sqlQuery(xf, "select * from samples.dbo.adult")
    > str(go1)
    data.frame':   32560 obs. of  15 variables:….
```

RODBC Package

This allow you to run any SQL command on the database

```
> sqlQuery(xf, "CREATE TABLE Rdemo (id INT IDENTITY,x1 float,x2 float)")
character(0)
> sqlColumns(xf,"RDemo")
```

	TABLE_CAT	TABLE_SCHEM	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME
1	Samples	dbo	RDemo	Id	4	int identity
2	Samples	dbo	RDemo	x1	6	float
3	Samples	dbo	RDemo	x2	6	float .....

RODBC Package

We can use R to run processes which are difficult or impossible in DBMS

Example: calculate lag values

```
> for (i in 1:10)
> {
> LagRDemo<-sqlQuery(xf,paste("SELECT * FROM Rdemo WHERE Id
    BETWEEN ",(100000*(i-1)-10)," AND ",(100000*i), " ORDER BY Id"))
> add.val<-(i!=1)*10
>LagRDemo=cbind(LagRDemo[(add.val+1):(add.val+100000),],lagDF(La
    gRDemo,10)[(add.val+1):(add.val+100000),2:3])
> sqlSave(xf, LagRDemo, append = (i!=1), rownames=F)
> }
```

RODBC Package

Example: calculate lag values

(local).Samples - SQLQuery2.sql* | (local).Samples - SQLQuery1.sql* | Summary

```
SELECT TOP 20 *
FROM dbo.LagRDemo
```

Results | Messages

	Id	x1	x2	lag10x1	lag10x2
1	1	0.197886184129313	2.60105392068939	NULL	NULL
2	2	0.132014468185215	4.75113066290347	NULL	NULL
3	3	0.276455774987912	5.06079939483775	NULL	NULL
4	4	0.26506516501074	3.25849801801553	NULL	NULL
5	5	0.418216819166843	1.45185764645281	NULL	NULL
6	6	0.891177251079599	1.02015967467461	NULL	NULL
7	7	0.139528880439235	3.30111929831258	NULL	NULL
8	8	0.165196107546509	0.696813236356887	NULL	NULL
9	9	0.677401174628238	4.93304380157239	NULL	NULL
10	10	0.751722983550462	0.046977113350154	NULL	NULL
11	11	0.0060936302676223	3.05075091243198	0.197886184129313	2.60105392068939
12	12	0.212356757572768	0.818362616887777	0.132014468185215	4.75113066290347
13	13	0.0014865794624265	8.31926255344835	0.276455774987912	5.06079939483775
14	14	0.875417620986968	9.85751624544488	0.26506516501074	3.25849801801553
15	15	0.39635685002424	2.63910831626234	0.418216819166843	1.45185764645281
16	16	0.0463949175158612	6.95738058874123	0.891177251079599	1.02015967467461
17	17	0.749901602460901	4.54760803663352	0.139528880439235	3.30111929831258
18	18	0.927696234686572	2.68423359706788	0.165196107546509	0.696813236356...
19	19	0.21656473897217	6.72300622030899	0.677401174628238	4.93304380157239
20	20	0.251915093392728	7.36397052231861	0.751722983550462	0.046977113350...

Biglm Package

Building Generalized linear models on big data

- Loading data into memory in chunks
- Processing the last chunk and updating the sufficient statistic required for the model
- Disposes the last chunk and loading the next chunk
- Repeats until end of file

Biglm Package

```
library(biglm)
make.data<-function(filename, chunksize,...){
    conn<-NULL
    function(reset=FALSE){
        if(reset){
        if(!is.null(conn)) close(conn)
        conn<<- file (description=filename, open="r")
        } else{
    rval<-read.csv(conn, nrows=chunksize,...)
    if (nrow(rval)==0) {
    close(conn)
    conn<<-NULL
    rval<-NULL
    }
return(rval)
}}}
```

Biglm Package

```
> airpoll<-make.data("c:\\Rdemo.txt",chunksize=100000
    ,colClasses = list ("numeric","numeric","numeric")
    ,col.names = c("Id","x1","x2"))
> lmRDemo <-bigglm(Id~x1+x2,data=airpoll)
>summary(lmRDemo)
Large data regression model: bigglm(Id ~ x1 + x2, data = airpoll)
Sample size =  1e+06
                Coef       (95%       CI)      SE     p
(Intercept) 499583.8466 498055.6924 501112.0007 764.0771 0.0000
x1            -603.1151  -2602.7075   1396.4774 999.7962 0.5464
x2             143.6304    -56.2982    343.5591  99.9643 0.1508
```

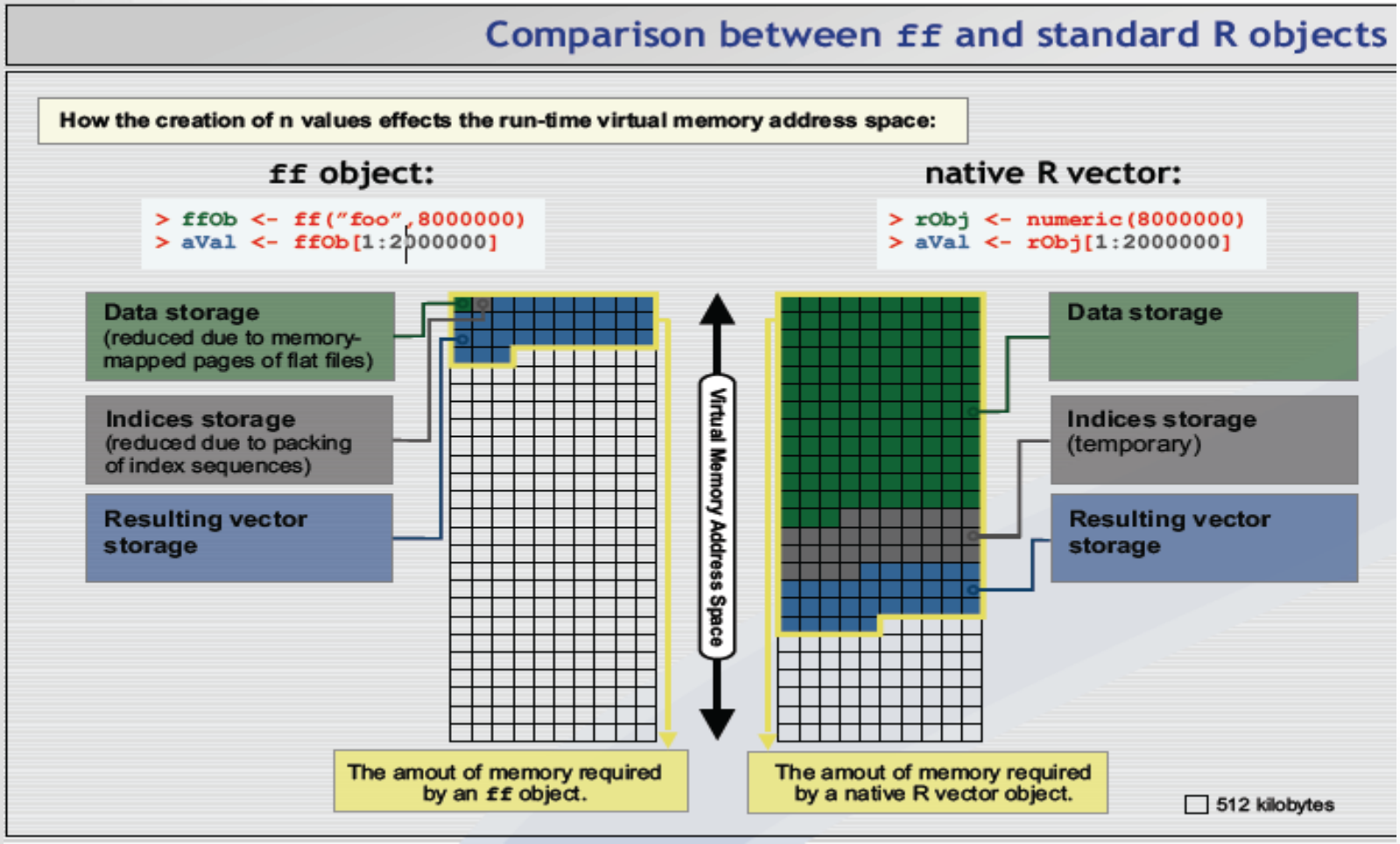
ff Package

- One of the main problems when dealing with large data set in R is memory limitations
  - On 32-bit OS the maximum amount of memory (i.e. virtual memory space) is limited to 2-4 GB
  - Therefore, one cannot store larger data into memory
  - It is impracticable to handle data that is larger than the available RAM for it drastically slows down performance.

ff Package

- The ff package offers file-based access to data sets that are too large to be loaded into memory, along with a number of higher-level functions.
- It provides Memory-efficient storage of large data on disk and fast access functions.
- The ff package provides data structures that are stored on disk but behave as if they were in RAM by transparently mapping only a section (pagesize) in main memory
- A solution to the memory limitation problem is given by considering only parts of the data at a time, i.e. instead of loading the entire data set into memory only chunks thereof are loaded upon request

ff Package



Source: www.r-project.org

How ff works

An example of what happens behind the scenes:

1 The first part of the code shows how an existing flat file is "opened": by calling `ff` a handle to a flat file is established

```
> tx <- ff("/tmp/texas_p", pagesize = 64*1024)
```

`tx` — handle to flat file

`ff` object

2 When extracting a subset of the data the commands do not differ from the "standard R procedure". However, the extraction procedure differs substantially

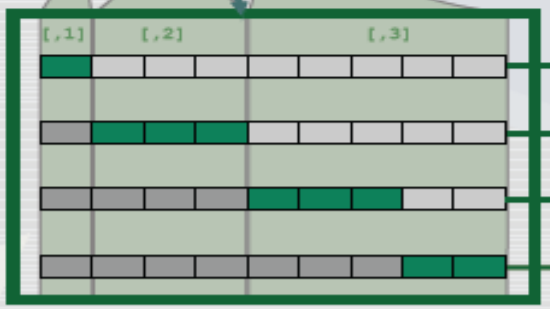
```
> ind <- c(1,9000,9005,9010,188417:196609)
> dat <- tx[ind]
```

`ind` — sequence transformation

	[,1]	[,2]	[,3]
from	1	9000	188417
to	1	9010	196609
by	0	5	1

`tx` — *FileMapper* *FileSection*

`dat`



1. The requested data item (at index 1) is in the currently active page.
2. The next three requested items are in the next page. The corresponding page is loaded and the items are extracted
3. The third block of requested items is split over two pages. The first page is mapped into main memory and the items are extracted
4. Then the next page containing the remaining entries is mapped and the corresponding items are extracted.

3 The object returned is a standard R object

```
> dat
 [1] 32  0  0 27  0 38 40  0  0  0  0  0 43 49 19
[16] 30 37 30 47 43 33 44 45 37 53 50 57 38 70 44
[31]  0  0  0  0  0 30 67 37 37 60 47 70 44 29 50
...
```

data entry (8 Bytes) — disk space not used

requested data — page (here: 64 KB)

ff Package

```
> library ( ff )
> N <- 1,000      # sample size #
> n <- 100        # chunk size #
> years <- 2000 : 2009
> types <- factor ( c ( " A " , " B " ," C " ) )

# Creating a ( one-dimensional ) flat file :

> Year <- ff ( years , vmode = 'ushort', length = N, update = FALSE ,
+ filename = " d:/tmp/Year.ff " , finalizer = "close" )

> Year
ff (open) ushort length=1000 (1000)
  [1]   [2]   [3]   [4]   [5]    [996] [997] [998] [999]  [1000]
    0     0     0     0     0 :     0     0     0     0       0
```

ff Package

Modifying data:

```
>  for ( i in chunk ( 1, N, n ) )
+  Year [ i ] <- sample ( years , sum ( i ) , TRUE )
```

```
> Year
ff (open) ushort length=1000 (1000)
  [1]     [2]     [3]     [4]      [996]  [997]  [998]   [999]  [1000]
 2001   2006   2007   2003   :   2002   2008   2007   2005   2003
```

ff Package

And the same for : Type

```
>  Type <- ff ( types , vmode = 'quad',  length = N, update  =FALSE ,
+  filename = " d:/tmp/Type.ff " , finalizer = "close" )

>  for ( i in chunk ( 1, N, n ) )
+  Type [ i ] <- sample ( types , sum ( i ) , TRUE )

> Type
ff (open) quad length=1000 (1000) levels:  A   B   C
  [1]   [2]   [3]   [4]      [996] [997] [998] [999]  [1000]
   A     A     B     B     :    C     C     B     A      C
```

ff Package

```
# create a data.frame #

> x <- ffdf ( Year = Year , Type = Type )
> x
ffdf (all open) dim=c(1000,2), dimorder=c(1,2) row.names=NULL
ffdf data
     Year Type
1    2001    A
2    2006    A
3    2007    B
4    2003    B
:       :    :
996  2002    C
997  2008    C
998  2007    B
999  2005    A
1000 2003    C
>
```

ff Package

- The data used:

  ASA 2009 Data Expo: Airline on-time performance

  http://stat-computing.org/dataexpo/2009/

- The data consisted of details of flight arrival and departure for all commercial flights within the USA, from October 1987 to April 2008.
- Nearly 120 million records, 29 variables (mostly integer-valued)

```
> x <- read.big.matrix("AirlineDataAllFormatted.csv", header=TRUE ,
+ type="integer", backingfile="airline.bin", descriptorfile="airline.desc",
+ extraCols="age")
```

ff Package

The challenge: find *min()* on extracted first column;

 # With ff:

> system.time(min(z[,1], na.rm=TRUE))

user system elapsed

2.188 1.360 **10.697**

> system.time(min(z[,1], na.rm=TRUE))

user system elapsed

1.504 0.820 **2.323**

 # With bigmemory:

> system.time(min(x[,1], na.rm=TRUE))

user system elapsed

1.224 1.556 **10.101**

> system.time(min(x[,1], na.rm=TRUE))

user system elapsed

1.016 0.988 **2.001**

ff Package

The challenge: random extractions

```
> theserows <- sample(nrow(x), 10000)
> thesecols <- sample(ncol(x), 10)
```

 # With ff:

```
> system.time(a <- z[theserows, thesecols])
```

user system elapsed

0.092 1.796 **60.574**

 # With bigmemory:

```
> system.time(a <- x[theserows, thesecols])
```

user system elapsed

0.020 1.612 **64.136**

bigmemory Package

- An R package which allows powerful and memory-efficient parallel analyses and data mining of massive data sets.
- Permits storing large objects (matrices etc.) in memory (on the RAM) using external pointer objects to refer to them.
- The data sets may also be file-backed, to easily manage and analyze data sets 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.martix , such that X is a pointer to the dataset that is saved in the RAM or on the hard drive.
- When a big.matrix, X , is passed as an argument to a function, it is essentially providing call-byreference rather than call-by-value behavior
- Backingfile is the root name for the file(s) for the cache of X.
- Descriptorfile the name of the file to hold the filebacked description, for subsequent use with attach.big.matrix
- attach.big.matrix creates a new big.matrix object which references previously allocated shared memory or file-backed matrices.

bigmemory Package

- The default big.matrix is not shared across processes and is limited to available RAM.
- A shared big.matrix has identical size constraints as the basic big.matrix, but may be shared across separate
- A file-backed big.matrix may exceed available RAM by using hard drive space, and may also be shared across processes.
  - big.matrix (nrow, ncol, type = “integer”, ….)
  - shared.big.matrix (nrow, ncol, type = “integer”, ….)
  - filedbacked.big.matrix (nrow, ncol, type = “integer”, ….)
  - read.big.matrix (filename, sep= , ….)

bigmemory Package

```
> library ( bigmemory)

# Creating A new BigMemory object

> X <- read.big.matrix ( “BigMem.csv” ,type = “double” , backingfile =
“BigMem.bin” , descriptorfile = “BigMem.desc”, shared = TRUE)
> X

An object of class “big.matrix”
Slot "address":
<pointer: 0x0196d058>
```

bigmemory Package

```
> X [ 1:4 , 1:4 ]
     [,1] [,2] [,3] [,4]
[1,]  100  200  300  400
[2,]  100  200  300  400
[3,]  100  200  300  400
[4,]  100  200  300  400
```

Creating an existing BigMemory object on a different machine

```
> Y <- attach.big.matrix ( "BigMem.desc" )

> Y

An object of class “big.matrix”

Slot "address":

<pointer: 0x01972b18>
```

bigmemory Package

```
> X [1,1] = 1111
> X [ 1:4 , 1:4 ]
     [,1] [,2] [,3] [,4]
[1,] 1111  200  300  400
[2,] 100   200  300  400
[3,] 100   200  300  400
[4,] 100   200  300  400
```

 # On different R:

```
> Y [ 1:4 , 1:4 ]
     [,1] [,2] [,3] [,4]
[1,] 1111  200  300  400
[2,] 100   200  300  400
[3,] 100   200  300  400
[4,] 100   200  300  400
```

bigmemory Package

```
> Z  <-  shared.big.matrix ( 1000 ,70, type = “double” )
> describe( Z )
$sharedType
[1] "SharedMemory"
$sharedName
[1] "d177ab0c-348c-484e-864f-53025015656e"
$nrow
[1] 1000
$ncol
[1] 70
$rowNames
NULL
$colNames
NULL
$type
[1] "double"
```

snow Package

- **S**imple **N**etwork **o**f **W**orkstations
- An R package which supports simple parallel computing.
- The package provides high-level interface for using a workstation cluster for parallel computations in R.

- Snow relies on the Master/Slave model of communcation:
- One device (master) controls one or more other devices (slaves)
- Note: communication is orders of magnitude slower than computation. For efficient parallel computing a dedicated high-speed network is needed.

snow Package

Starting and Stopping clusters:

The way to Initialize slave R processes depends on system configuration, for example:

```
>  cl <- makeCluster( 2, type = “SOCK" )
```

Shut down the cluster and clean up any remaining connections between machines:

```
> stopCluster ( cl )
```

snow Package

clusterCall (cl, fun , ...)

clusterCall calls a specified function with identical arguments on each node in the cluster.

The arguments to clusterCall are evaluated on the master, their values transmitted to the slave nodes which execute the function call.

```
> myfunc <- function (  x ) { x + 1 }
> myfunc_argument <- 5
> clusterCall ( cl, myfunc, myfunc_argument )
    [[1]]
    [1] 6

   [[2]]
    [1] 6
```

snow Package

Example: simulate random numbers

```
> clusterApply(cl,c(4,2),runif)
[[1]]
0.90365799 0.03189395 0.59713787 0.33039294 [1]
[[2]]
0.6620030 0.8329455 [1]
> system.time(clusterApply(cl,c(6000000,9000000),runif))
 user  system elapsed
  5.03    0.94   11.31
> system.time(runif(15000000))
user  system elapsed
5.11    0.12    5.47
```