

Scalable Strategies for Computing with Massive Data: The Bigmemory Project

Jay Emerson¹ and Mike Kane²
<http://www.bigmemory.org/>

(1) <http://www.stat.yale.edu/~jay/>, @johnwemerson
Associate Professor of Statistics, Yale University

(2) Research Faculty, Yale Center for Analytical Science

This talk also acknowledges work by Dan Adler, Christian Gleser, Oleg Nenadic, Jens Oehlschlegel, and Walter Zucchini (authors of package **ff**); Roger Peng (author of package **filehash**); Steve Weston and Revolution Analytics (authors of package **foreach** and most of the associated parallel backends); Thomas Lumley (author of **biglm**); Baglama and Reichel (authors of a 2005 paper that I'll reference); Bryan Lewis (author of **doRedis** and **irlba**); and many of the participants of VLDS 2011.

A new era

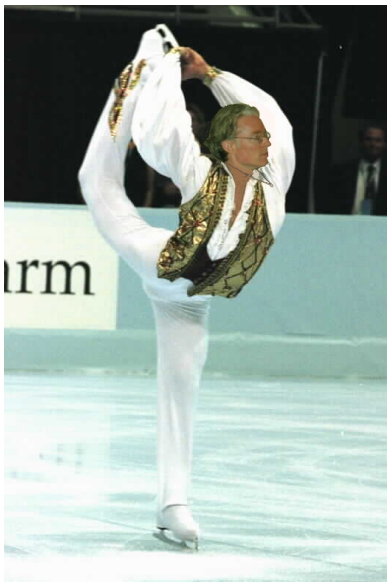
- “We have entered an era of massive scientific data collection, with a demand for answers to large-scale inference problems that lie beyond the scope of classical statistics.” – Efron (2005)
- “classical statistics” should include “mainstream computational statistics.” – Kane, Emerson, and Weston (in submission)

- Scalability
- Cost of computing is an important consideration
 - Cost of coding needs to be considered
 - Let's not keep reinventing the wheel if we can avoid it: some algorithms could work on both large and small problems. Others can't. Whether the methods should be used is a different question (and a good one).

- 1-2 TB of RAM? Must be nice!
 - I note that in the afternoon discussion Robert noted that resource constraints are an important consideration for many researchers, however.
- Avoid many languages. This message was brought to you by the letter 'R'.
- Open development
- Quality assessment
- Data processing and data reduction (and don't forget to *think*)...
 - Processed data may often be stored as a matrix

- Commented on streaming data and the importance of preprocessing
 - Reduced data often have a nice structure (e.g. like a matrix, say)
- Importance of Twitter tags...
 - mine is @johnwemerson, not as cool as @statpumpkin

I've only used Twitter to announce analyses of figure skating results!



- Important “big picture” issues:
 - How do we do this?
 - Why do we do this?

Terry Speed and Rafa Irizarry

- Not plastics, not statistics... controls!
- The hope that what we do has broader impact than our particular problem
- Rafa via Terry: “It’s not how big your data set is, it’s what you do with it that counts.”

- How big is big?
 - Moving target
 - Relative, not simply absolute
 - Gentleman's "open development" – extend to "open research"?
 - The Netflix competition

- “Why do you want to do that?”
- Statistics – Mathematics
- Manifolds
- `grep *Euclidean MarronSlideText.txt | more`
- Encouraged by Steve’s talk, I added some material to my own talk earlier today: restarted Lanczos bidiagonalization methods, augmentation of Krylov subspaces, and certain Ritz or harmonic Ritz vectors.

A tip of the hat: **ff** and **filehash**

filehash: a simple key-value database in pure R

ff has some very cool features (differentiating itself from **bigmemory**):

- support for arrays and data.frames, not “just” matrices
- additional types (quad, nibble, ...) and extended types (factors, POSIXct, and user-defined)
- cool indexing for improved random access performance
- fast filtering of data frames via package **bit**

A tip of the hat: **foreach**

Along with optional parallel packends **doMC**, **doSNOW**, **doMPI**, **doRedis**, and **doSMP**, package **foreach** provides an elegant framework for portable parallel programming.

```
> library(foreach)
>
> library(doMC)
> registerDoMC(2)
>
> ans <- foreach(i = 1:10, .combine = c) %dopar%
+   {
+     i^2
+   }
>
> ans
[1] 1 4 9 16 25 36 49 64 81 100
```

Example data sets

- Airline on-time data

- 2009 JSM Data Expo (thanks, Hadley!)
- About 120 million commercial US airline flights over 20 years
- 29 variables, integer-valued or categorical (recoded as integer)
- About 12 gigabytes (GB)
- <http://stat-computing.org/dataexpo/2009/>

- Netflix data

- About 100 million ratings from 500,000 customers for 17,000 movies
- About 2 GB stored as integers
- No statisticians on the winning team; hard to find statisticians on the leaderboard
- Top teams: access to expensive hardware; professional computer science and programming expertise
- <http://www.netflixprize.com/>

Why R?

R is the *lingua franca* of statistics (thanks to John Chambers for getting the ball rolling with the S language; Robert Gentleman, Ross Ihaka, the R Core team and the multitude of contributors for the R language and community).

- The syntax is simple and well-suited for data exploration and analysis.
- It has excellent graphical capabilities.
- It is extensible, with over 2500 packages available on CRAN alone.
- It is open source and freely available for Windows/MacOS/Linux platforms.

In a nutshell...

- The approaches adopted by statisticians in analyzing small data sets don't (often) scale to massive ones.
- Statisticians who want to explore massive data must
 - be aware of the various pitfalls;
 - adopt new approaches to avoid them.

Importing and managing massive data

- `read.table()` and R objects:
 - memory overhead as much as 100% of size of data
 - `data.frame` and `matrix` objects not available in shared memory
 - limited in size by available RAM, recommended maximum 10%-20% of RAM

- `read.big.matrix()` and `big.matrix` objects:
 - matrix-like data (not data frames), compatible with linear algebra libraries
 - no memory overhead
 - supports shared memory for efficient parallel programming
 - supports file-backed objects for data larger-than-RAM
 - leverages the Boost C++ libraries

- Databases and other alternatives:
 - Slower performance (often), no formal shared memory
 - Not compatible with linear algebra libraries
 - Require customized coding (chunking algorithms, generally)

Importing and managing massive data

With the full Airline data (~ 12 GB), **bigmemory**'s file-backing allows you to work with the data even with only 4 GB of RAM, for example:

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

An object of class "big.matrix"

Slot "address":

<pointer: 0x3031fc0>

```
> rm(x)
```

```
> x <- attach.big.matrix("airline.desc")
```

```
> dim(x)
```

```
[1] 123534969      29
```

Exploring massive data

```
> summary(x[, "DepDelay"])
      Min.      1st Qu.      Median      Mean
-1410.000    -2.000      0.000     8.171
      3rd Qu.      Max.      NA's
      6.000    2601.000 2302136.000

>
> quantile(x[, "DepDelay"],
+         probs=c(0.5, 0.9, 0.99), na.rm=TRUE)
50% 90% 99%
  0  27 128
```

Example: caching in action

In a fresh R session on this laptop, newly rebooted:

```
> library(bigmemory)
> library(biganalytics)
> xdesc <- dget("airline.desc")
> x <- attach.big.matrix(xdesc)
> system.time( numplanes <- colmax(x, "TailNum",
+                               na.rm=TRUE) )
  user  system elapsed
0.770   0.550   6.144
> system.time( numplanes <- colmax(x, "TailNum",
+                               na.rm=TRUE) )
  user  system elapsed
0.320   0.000   0.355
```

Split-apply-combine

- Many computational problems in statistics are solved by performing the same calculation repeatedly on independent sets of data. These problems can be solved by
 - partitioning the data (the *split*)
 - performing a single calculation on each partition (the *apply*)
 - returning the results in a specified format (the *combine*)
- Recent attention: it can be particularly efficient and easily lends itself to parallel computing
- “split-apply-combine” was coined by Hadley Wickham, but the approach has been supported on a number of different environments for some time under different names:
 - SAS: *by*
 - Google: MapReduce
 - Apache: Hadoop

Exploring massive data

```
> GetDepQuantiles <- function(rows, data) {  
+   return(quantile(data[rows, "DepDelay"],  
+                   probs=c(0.5, 0.9, 0.99), na.rm=TRUE))  
+ }  
>  
> groups <- split(1:nrow(x), x[, "DayOfWeek"])  
>  
> qs <- sapply(groups, GetDepQuantiles, data=x)  
>  
> colnames(qs) <- c("Mon", "Tue", "Wed", "Thu",  
+                  "Fri", "Sat", "Sun")  
> qs
```

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
50%	0	0	0	0	0	0	0
90%	25	23	25	30	33	23	27
99%	127	119	125	136	141	116	130

Aside: foreach

The user may register any one of several “parallel backends” like **doMC**, or none at all. The code will either run sequentially or will make use of the parallel backend, without modification.

```
> library(foreach)
>
> library(doMC)
> registerDoMC(2)
>
> ans <- foreach(i = 1:10, .combine = c) %dopar%
+   {
+     i^2
+   }
>
> ans
[1] 1 4 9 16 25 36 49 64 81 100
```

Concurrent programming with **foreach** (1995 only)

- Such split-apply-combine problems can be done in parallel.
- Here, we start with one year of data only, 1995.
- Why only 1995? Memory implications.
- The message: shared memory is essential.

```
> library(foreach)
> library(doSNOW)
> cl <- makeSOCKcluster(4)
> registerDoSNOW(cl)
>
> x <- read.csv("1995.csv")
> dim(x)
[1] 5327435      29
```

Concurrent programming with **foreach** (1995 only)

4 cores: 37 seconds (memory overhead > 2.5 GB)

3 cores: 23 seconds

2 cores: 15 seconds

1 core: 8 seconds

```
> groups <- split(1:nrow(x), x[, "DayOfWeek"])
>
> qs <- foreach(g=groups, .combine=rbind) %dopar% {
+   GetDepQuantiles(g, data=x)
+ }
>
> # Same result as before
```

Challenge: find plane “birthdays”

```
> birthmonth <- function(y) {  
+   minYear <- min(y[, 'Year'], na.rm=TRUE)  
+   these <- which(y[, 'Year']==minYear)  
+   minMonth <- min(y[these, 'Month'], na.rm=TRUE)  
+   return(12*minYear + minMonth)  
+ }  
>  
> time.0 <- system.time( {  
+   planemap <- split(1:nrow(x), x[, "TailNum"])  
+   planeStart <- sapply( planemap,  
+     function(i) birthmonth(x[i, c('Year', 'Month')],  
+                               drop=FALSE)) )  
+ } )  
>  
> time.0  
      user  system elapsed  
53.520    2.020   78.925
```

Parallel split-apply-combine

Using 4 cores, we can reduce the time to ~ 20 seconds (not including the `read.big.matrix()`, repeated here for a special reason:

```
x <- read.big.matrix("airline.csv", header = TRUE,
                    backingfile = "airline.bin",
                    descriptorfile = "airline.desc",
                    type = "integer",
                    extraCols = "Age")
planeindices <- split(1:nrow(x), x[, "TailNum"])
planeStart <- foreach(i = planeindices,
                      .combine = c) %dopar% {
  birthmonth(x[i, c("Year", "Month"), drop = FALSE])
}
x[, "Age"] <- x[, "Year"] * as.integer(12) +
             x[, "Month"] -
             as.integer(planeStart[x[, "TailNum"]])
```

Massive linear models via **biglm**

```
> library (biganalytics)
> x <- attach.big.matrix("airline.desc")
> blm <- biglm.big.matrix(DepDelay ~ Age, data = x)
> summary(blm)
```

Large data regression model: `biglm(formula = formula, data = data , ...)`

Sample size = 84406323

	Coef	(95%	CI)	SE	p
(Intercept)	8.5889	8.5786	8.5991	0.0051	0
Age	0.0053	0.0051	0.0055	0.0001	0

irlba: Implicitly-restarted Lanczos bidiagonalization algorithm

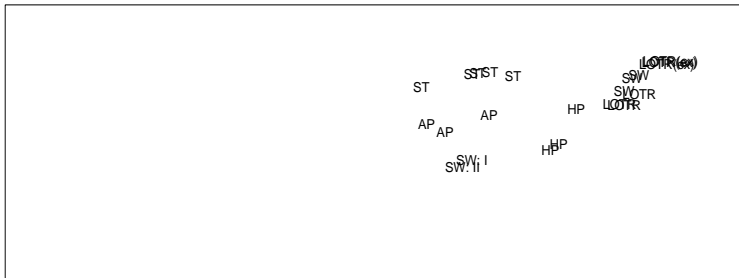
Implemented by Bryan Lewis.

- The implicitly restarted Lanczos bi-diagonalization (IRLBA) algorithm finds a few approximate singular values and corresponding singular vectors of a matrix using the augmented implicitly-restarted Lanczos bidiagonalization method of Baglama and Reichel. It is a fast and memory-efficient way to compute a partial SVD.
- “Augmented Implicitly Restarted Lanczos Bidiagonalization Methods”, J. Baglama and L. Reichel, SIAM J. Sci. Comput. 2005.

irlba: Code snippet for making an important point about **bigmemory**

```
`irlba` <-  
  function(A, nu=5, nv=5, adjust=3, aug="ritz",  
          sigma="ls", maxit=1000, m_b=20,  
          reorth=1, tol=1e-6, V=NULL)  
{  
  ...  
  
  for (i in 1:nrow(A)) {  
    W[i,j] = sum(A[i,] * V[,j,drop=FALSE])  
  }  
  
  ...  
}
```

Netflix: a PCA (not necessarily a great one, but...)



The Bigmemory Project proposes three new ways to work with very large sets of data:

- memory and file-mapped data structures, which provide access to arbitrarily large sets of data while retaining a look and feel that is familiar to statisticians;
- data structures that are shared across processor cores on a single computer, in order to support efficient parallel computing techniques when multiple processors are used;
- and file-mapped data structures that allow concurrent access by the different nodes in a cluster of computers.

Even though these three techniques are currently implemented only for R, they are intended to provide a flexible framework for future developments in the field of statistical computing.

A plug for JSM (making the most of Miami in August??!!)

Session 101: The Future of Statistical Computing Environments

- Luke Tierney: Some Developments for the R Engine
- Simon Urbanek: Taking Statistical Computing Beyond S and R
- Andrew Runnals: CXXR: an Ideas Hatchery for Future R Development
- Mike Kane: The Q Project: Explorations Using the Parrot Virtual Machine

- Dirk Eddebuettel, Bryan Lewis, Steve Weston, and Martin Schultz, for their feedback and advice over the last three years
- Bell Laboratories (Rick Becker, John Chambers and Allan Wilks), for development of the S language
- Ross Ihaka and Robert Gentleman, for their work and unselfish vision for R
- The R Core team
- David Pollard, for pushing us to better communicate the contributions of the project to statisticians
- John Hartigan, for years of teaching and mentoring
- John Emerson (my father, Middlebury College), for getting me started in statistics and pushing me to learn to write code
- Many of my students, for their willingness argue with me

Extra: Extending capabilities versus extending the language

- Extending R's capabilities:
 - Providing a new algorithm, statistical analysis, or data structure
 - Examples: `lm()`, or package **bcp**
 - Most of the 2500+ packages on CRAN and elsewhere

- Extending the language:
 - Example: **grDevices**, a low-level interface to create and manipulate graphics devices
 - Example: **grid** graphics
 - The packages of the Bigmemory Project:
 - a developer's interface to underlying operating system functionality which could not have been written in R itself
 - a higher-level interface designed to mimic R's matrix objects so that statisticians can use their current knowledge of computing with data sets as a starting point for dealing with massive ones.