The Bigmemory Project

Michael J. Kane and John W. Emerson
Yale University
April 29, 2010

Multi-gigabyte data sets often challenge and frustrate **R** users. **C/C++** programming can provide efficiencies, but is cumbersome for interactive data analysis and lacks the flexibility and power of **R**'s rich statistical programming environment. The package ***bigmemory*** and sister packages ***biganalytics***, ***synchronicity***, ***bigtabulate***, and ***bigalgebra*** bridge this gap, implementing massive matrices and supporting their manipulation and exploration. The data structures may be allocated to shared memory, allowing separate processes on the same computer to share access to a single copy of the data set. The data structures may also be file-backed, allowing users to easily manage and analyze data sets larger than available RAM and share them across nodes of a cluster. These features of the Bigmemory Project open the door for powerful and memory-efficient parallel analyses and data mining of massive data sets, even on modest hardware.

Introductory Example: the 2009 JSM Data Expo

Consider the complete airline on-time performance data from the 2009 JSM Data Expo. The processed data set, `airline.csv`, is approximately 11 GB (about 120 million rows and 29 columns) with factors coded as integers (see `http://www.bigmemory.org/` for processing information). The `read.big.matrix()` call creates the binary file-backing `airline.bin` associated with the `big.matrix` object `x`. Subsequent **R** sessions can attach instantly to `airline.bin` without incurring the one-time overhead (about 25 minutes) associated with creating the backing. A summary of the entire data set is easily obtained using the new `summary()` method. Note the surprising presence of negative arrival and departure delays: exploratory data analysis in action via ***bigmemory***. The summary only takes 3-4 minutes to process the 11 GB of data on a laptop with only 4 GB of RAM.

```
library(bigmemory)
library(biganalytics)
x <- read.big.matrix("airline.csv", type="integer", header=TRUE,
                      backingfile="airline.bin",
                      descriptorfile="airline.desc",
                      extraCols="Age")
summary(x)

#                          min        max        mean       NA's
#Year                     1987       2008     1998.62          0
#Month                       1         12        6.55          0
#DayofMonth                  1         31       15.72          0
#DayOfWeek                   1          7        3.94          0
#ArrDelay                -1437       2598        7.05    2587529
```

```
#DepDelay                    -1410        2601          8.17      2302136
#... (other variables omitted here) ...
```

Overview

Data frames and matrices in **R** are easy to use, with typical manipulations executing quickly on data sets much smaller than available RAM. They suit the needs of many **R** users and work seamlessly with existing **R** functions and packages. However, problems arise with larger data sets and when increased memory requirements of parallel programming strain the system.

The Bigmemory Project offers packages for two purposes. First, ***bigmemory***, ***biganalytics***, and ***bigtabulate*** have been designed to provide a minimalist, elegant framework for users to manage and explore large data sets, even on modest hardware (expensive workstations or clusters are not required). The interface is designed to mimic **R**'s familiar `matrix` syntax. Matthew Keller, Assistant Professor of Psychology, University of Colorado at Boulder offered the following testimonial about ***bigmemory***: "I love that it's intuitive and doesn't require a lot of learning new ways to code things."

Second, the packages of the Bigmemory Project provide a foundation for memory-efficient parallel programming and can serve as building blocks for developers of new high-performance computing tools in **R**. When used in conjunction with a parallel package (such as ***foreach***, ***snow***, ***Rmpi***, or ***multicore***, for example), even shared-memory parallel-computing becomes accessible to non-experts. The programming interface is stable, and offers the flexibility to support the development of algorithms working seamlessly on both `big.matrix` and traditional `matrix` objects. For examples of this, look first at the function `mwhich()`; it offers flexible `which()`-like functionality that is computationally efficient and avoids memory overhead. In addition, all the functions provided by ***bigtabulate*** may be used with `matrix` and `big.matrix` objects alike.

Underneath the Hood of the Bigmemory Project

The packages of the Bigmemory Project use the Boost Interprocess **C++** library to provide platform-independent support for massive matrices that may be shared across **R** processes. Innovative use of **C++** accessors supports matrices of `double`, `integer`, `short`, and `char`, as well as the development of algorithms working seamlessly on `big.matrix` objects or traditional **R** matrices.

Example: Airplane Ages and Parallel Processing

We would like to approximate the age of each plane at the time of each flight. This first requires calculation of an approximate "birthmonth" for each plane: the month of the first appearance in the data set. Given a matrix `y` containing `Year` and `Month` for all flights of a given plane, `birthmonth(y)` returns the month (in months AD) of the earliest flight:

```
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 - 1)
}
```

A traditional approach to calculating all the birthmonths might use a `for()` loop:

```
allplanes <- unique(x[,'TailNum'])
planeStart <- rep(0, length(allplanes))
for (i in allplanes) {
  planeStart[i] <- birthmonth( x[mwhich(x, 'TailNum', i, 'eq'),
                             c('Year', 'Month'), drop=FALSE] )
}
```

With about 13,000 flights this takes about 9 hours, even with the relative fast and memory-efficient use of `mwhich()`.

A far more efficient alternative is to first obtain a list of row indices for each plane:

```
library(bigtabulate)
planeindices <- bigsplit(x, 'TailNum')
```

Here, the use of the new function `bigsplit()` is equivalent to

```
planeindices <- split(1:nrow(x), x[,'TailNum'])
```

but is faster (16 versus 29 seconds) and more memory efficient (with peak memory usage of 2 versus 3 GB). Either way, `planeindices[i]` contains all row indices corresponding to flights with `TailNum` equal to `i`. This requires several hundred MB, but is computationally more efficient in this problem. For example, `planeindices` may be used with `sapply()` in the obvious way, completing the task in a mere 30 seconds:

```
planeStart <- sapply(planeindices,
                  function(i) birthmonth(x[i, c('Year','Month'),
                                           drop=FALSE]))
```

The looping structure `foreach()` of package ***foreach*** can be a powerful and flexible alternative to `for()` or functions like `lapply()` and `sapply()`. It can also take advantage of the shared-memory capability of ***bigmemory***. Package ***doMC*** provides one of several available "parallel backends" for the function `foreach()`, allowing the work to be automatically distributed to available processor cores:

```
library(doMC)
registerDoMC(cores=2)
planeStart <- foreach(i=planeindices, .combine=c) %dopar% {
  return(birthmonth(x[i, c('Year','Month'), drop=FALSE]))
}
```

The syntax of a `foreach()` loop is slightly different from the syntax of a traditional loop, but its benefits are clear: in this example, it takes only 14 seconds to calculate the plane birthmonths using two processor cores.[1] Both cores share access to the same master copy of the airline data (with `Year` and `Month` cached in RAM); individual calls to `birthmonth()` are relatively small in size. Without the `registerDoMC()` initialization, the `foreach()` loop would run on a single processor core, much like `sapply()`, but taking about 24 seconds in this problem with lower memory overhead than `sapply()`.

Finally, the plane ages at the time of all flights may be calculated:

```
x[,'Age'] <- x[,'Year']*as.integer(12) +
          x[,'Month'] - as.integer(planeStart[x[,'TailNum']])
```

This arithmetic is conducted on **R** vectors extracted from the `big.matrix`; use of `as.integer()` helps keep the memory consumption under control.

Concluding Example: a Big Regression

In addition to providing basic functions for exploratory data analysis, the package ***biganalytics*** provides a wrapper for Thomas Lumley's ***biglm*** package, supporting massive linear and generalized linear models.[2] The following toy example examines the airline arrival delays as a linear function of the age of the plane at the time of the flight and the year of the flight. About 85 million flights are used (because of missing airplane tailcodes). We estimate that use of **R**'s `lm()` function would require more than 10 GB of RAM of memory overhead, while this example runs in about 3 minutes with only several hundred MB of memory overhead.

```
blm <- biglm.big.matrix(ArrDelay ~ Age + Year, data=x)
summary(blm)
```

```
#Large data regression model: biglm(formula = formula, data = data, ...)
#Sample size = 84216580
#                Coef    (95%     CI)     SE p
#(Intercept) 91.6149 87.6509 95.5789 1.9820 0
#Age          0.0144  0.0142  0.0146 0.0001 0
#Year        -0.0424 -0.0444 -0.0404 0.0010 0
```

From this, we might conclude that older planes are associated with increased predicted delays, and predicted delays in recent years are lower. However, this exercise is merely for illustrative purposes; a serious study of airline delays would quickly reject this oversimplification and discover problems with this particular regression.

Additional Information and Supporting Material

These examples were tested both in Linux 64-bit and Windows 7 Enterprise 64-bit

[1]We should note that ***doMC*** and ***multicore*** are particularly well-suited for this. When other parallel backends are used, one additional command is required in the `birthmonth()` function: `x <- attach.big.matrix(xdesc)` where `xdesc <- describe(x)` would be required just prior to the `foreach()` loop, providing explicit shared-memory access across processes. In contrast, `multicore` automatically operates on shared memory, avoiding the need for this extra step.

[2]Package ***biganalytics*** also provides `bigkmeans()`, and other analytics may be added to the package in the future.

environments. Older versions of Windows operating systems (including Vista 64-bit) seem to suffer from extremely inefficient caching behavior with filebackings and are not recommended for use with ***bigmemory***; 32-bit environments will be limited by approximately 2 GB of addressable memory.

The packages are available via R-Forge and on CRAN as of late April, 2010; please see `http://www.bigmemory.org/` for more information. There is a short vignette available in the Documentation area, as well as presentation slides introducing ***bigmemory*** and providing some benchmarks and shared-memory parallel programming examples. Please do not use the older version of ***bigmemory*** archived on CRAN (versions <= 3.12).

Citations

1. 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.

2. 2009 JSM Data Expo: Airline on-time performance. `http://stat-computing.org/dataexpo/2009/`.

3. Thomas Lumley (2009). ***biglm***: bounded memory linear and generalized linear models. **R** package version 0.7, `http://CRAN.R-project.org/package=biglm`.

4. **R** Development Core Team (2009). **R**: A language and environment for statistical computing. **R** Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, `http://www.R-project.org`.

5. Luke Tierney, A. J. Rossini, Na Li and H. Sevcikova (). ***snow***: Simple Network of Workstations. **R** package version 0.3-3, `http://CRAN.R-project.org/package=snow`.

6. Simon Urbanek (2009). ***multicore***: Parallel processing of **R** code on machines with multiple cores or CPUs. **R** package version 0.1-3, `http://www.rforge.net/multicore/`.

7. Stephen Weston and REvolution Computing (2009). ***doMC***: Foreach parallel adaptor for the ***multicore*** package. **R** package version 1.2.0, `http://CRAN.R-project.org/package=doMC`.

8. Stephen Weston and REvolution Computing (2009). ***foreach***: Foreach looping construct for **R**. **R** package version 1.3.0, `http://CRAN.R-project.org/package=foreach`.

9. Hao Yu (2010). ***Rmpi***: Interface (Wrapper) to MPI (Message-Passing Interface). **R** package version 0.5-8, `http://www.stats.uwo.ca/faculty/yu/Rmpi`.