DS432 - Regression 2

Saha Debanshee Gopal

11/07/2016

The Need for Big Data?

Statistical softwares like Base R and SAS reads data by default into your memory(RAM).It's easy to exhaust RAM by storing unnecessary information. Our OS and architecture can access only 4GB of data of memory on 32 bit system. And over dumping of data can slow down your systems.

Hence we need robust systems to handle big data. Our systems should be able to tackle data in terms of Volume, Variety and Velocity of the Data.

The Volume of data is continuously growing. We have huge feeds of data being generated everyday e.g. from social networking sites like your twitter ,Facebook,etc.So Data storage and Data Processing is the need of the hour. Also your conventional databases like Teradata, SQL server, etc. cannot handle huge Volumes of data (Terabytes of data).Velocity signifies the speed with which we can handle data. Another typical problem which data analysts are facing today is to load and analyse unstructured forms of data like text, graphics, etc.

So different kinds of Big Data systems are being designed to handle all these glitches present in our traditional systems.

Data generation :

450 MB File

n1 <- 10000000 x1 <- 1:n1 x2 <-runif(n1,5,95) x3 <- rbinom(n1,1,.4) x4 <- rnorm(n1, mean=-30, sd=200) x5 <- runif(n1,-5000,5000) b0 <- 17; b1 <- -0.466; b2 <- 0.037; b3 <- -5.2; b4 <- 2; b5 <- 0.00876 sigma <- 1.4 epsilon <- rnorm(x1,0,sigma) y <- b0 + b1$x1$ + $b2$x2 + b3$x3$ + $b4$x4 + b5$x5$ + $x1$x2 + epsilon data14a<-cbind(y,x1,x2,x3,x4,x5)

4.5 GB File

n1 <- 50000000 x1 <- 1:n1 x2 <-runif(n1,5,95) x3 <- rbinom(n1,1,.4) x4 <- rnorm(n1, mean=-30, sd=200) x5 <- runif(n1,-5000,5000) b0 <- 17; b1 <- -0.466; b2 <- 0.037; b3 <- -5.2; b4 <- 2; b5 <- 0.00876 sigma <- 1.4 epsilon <- rnorm(x1,0,sigma) y <- b0 + b1$x1$ + $b2$x2 + b3$x3$ + $b4$x4 + b5$x5$ + $x1$x2 + epsilon data14a<-cbind(y,x1,x2,x3,x4,x5)

```
d1 <- read.csv("~/Documents/data_small.txt",sep = "")
```

LM :

```
lm1 <- lm(y~x1+x2+x3+x4+x5,data=d1)
summary(lm1)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2 + x3 + x4 + x5, data = d1)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -224806750  -42015477       5298   41975482  224834062
##
## Coefficients:
##               Estimate Std. Error   t value Pr(>|t|)
## (Intercept) -2.500e+08  6.872e+04 -3637.631   <2e-16 ***
## x1           4.953e+01  8.217e-03  6028.175   <2e-16 ***
## x2           4.999e+06  9.130e+02  5475.981   <2e-16 ***
## x3          -3.619e+04  4.842e+04    -0.747    0.455
## x4           1.105e+01  1.186e+02     0.093    0.926
## x5          -8.168e+00  8.218e+00    -0.994    0.320
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 75010000 on 9999994 degrees of freedom
## Multiple R-squared:  0.869,  Adjusted R-squared:  0.869
## F-statistic: 1.326e+07 on 5 and 9999994 DF,  p-value: < 2.2e-16
```

Conventional lm() and glm() functions often fail when dealing with large datasets because of the limited availability of memory lm will work for 885 mb file but won't work for 4.5 Gb file.

BIGLM :

```
library(biglm)
```

```
## Loading required package: DBI
```

```
mod1 <- biglm(y~x1+x2+x3+x4+x5,data = d1)
summary(mod1)
```

```
## Large data regression model: biglm(y ~ x1 + x2 + x3 + x4 + x5, data = d1)
## Sample size =  10000000
##                       Coef           (95%           CI)          SE      p
## (Intercept) -2.499684e+08 -2.501058e+08 -2.498310e+08 68717.3599 0.0000
## x1           4.953410e+01  4.951760e+01  4.955050e+01     0.0082 0.0000
## x2           4.999437e+06  4.997611e+06  5.001263e+06   912.9757 0.0000
## x3          -3.618826e+04 -1.330310e+05  6.065448e+04 48421.3672 0.4548
## x4           1.104910e+01 -2.261006e+02  2.481987e+02   118.5748 0.9258
## x5          -8.167900e+00 -2.460320e+01  8.267400e+00     8.2176 0.3202
```

```
mod2 <- bigglm(y~x1+x2+x3+x4+x5,data = d1)
summary(mod2)
```

```
## Large data regression model: bigglm(y ~ x1 + x2 + x3 + x4 + x5, data = d1)
## Sample size =  1e+07
##                       Coef           (95%           CI)          SE      p
## (Intercept) -2.499684e+08 -2.501058e+08 -2.498310e+08 68717.3599 0.0000
## x1           4.953410e+01  4.951760e+01  4.955050e+01     0.0082 0.0000
## x2           4.999437e+06  4.997611e+06  5.001263e+06   912.9757 0.0000
```

```
## x3          -3.618826e+04 -1.330310e+05  6.065448e+04 48421.3672 0.4548
## x4           1.104910e+01 -2.261006e+02  2.481987e+02   118.5748 0.9258
## x5          -8.167900e+00 -2.460320e+01  8.267400e+00     8.2176 0.3202
```

```
family(mod2)
```

```
##
## Family: gaussian
## Link function: identity
```

```
deviance(mod2)
```

```
## [1] 5.626716e+22
```

```
AIC(mod2,k=2)
```

```
## [1] 5.626716e+22
```

```
sdd<-summary(mod2)$sigma
```

biglm() and bigglm(), from the package biglm by Thomas Lumley. They fit LMs and GLMs using an amount of memory which is only O(p^2),where p is the number of covariates.

```
library(hier.part)
```

```
## Loading required package: gtools
```

```
env <- d1[,2:6]
all.regs(d1$y, env, fam = "gaussian", gof = "Rsqu",
         print.vars = TRUE)
```

```
## regressions done: formatting results
```

```
##    variable.combination          gof
## 1                 Theta 0.000000e+00
## 2                    x1  4.760399e-01
## 3                    x2  3.927998e-01
## 4                    x3  6.353848e-11
## 5                    x4  1.466534e-09
## 6                    x5  1.455615e-08
## 7                 x1 x2  8.689654e-01
## 8                 x1 x3  4.760400e-01
## 9                 x1 x4  4.760399e-01
## 10                x1 x5  4.760399e-01
## 11                x2 x3  3.927999e-01
## 12                x2 x4  3.927998e-01
## 13                x2 x5  3.927998e-01
## 14                x3 x4  1.530030e-09
## 15                x3 x5  1.461921e-08
## 16                x4 x5  1.602212e-08
## 17             x1 x2 x3  8.689654e-01
## 18             x1 x2 x4  8.689654e-01
## 19             x1 x2 x5  8.689654e-01
## 20             x1 x3 x4  4.760400e-01
## 21             x1 x3 x5  4.760400e-01
## 22             x1 x4 x5  4.760399e-01
## 23             x2 x3 x4  3.927999e-01
## 24             x2 x3 x5  3.927999e-01
## 25             x2 x4 x5  3.927998e-01
```

```
## 26            x3 x4 x5  1.608514e-08
## 27         x1 x2 x3 x4  8.689654e-01
## 28         x1 x2 x3 x5  8.689654e-01
## 29         x1 x2 x4 x5  8.689654e-01
## 30         x1 x3 x4 x5  4.760400e-01
## 31         x2 x3 x4 x5  3.927999e-01
## 32      x1 x2 x3 x4 x5  8.689654e-01
```

BIG MEMORY/BIG ANALYTICS

This package consists of:

big.matrix is an R object that simply points to a data structure in C++. Local to a single R process and is limited by available RAM.

shared.big.matrix is similar, but can be shared among multiple R processes (similar to parallelism on data)

filebacked.big.matrix does not point to a data structure; instead it points to a file on disk containing the matrix, and the file can be shared across a cluster The major advantages of using this package is:

Can store a matrix in memory, restart R, and gain access to the matrix without reloading data. Great for big data. Can share the matrix among multiple R instances or sessions. Access is fast because RAM is fast. C++ also helps. One ***disadvantage*** could be that that matrices contain only one type of data.

```
library(bigmemory)
```

```
## Loading required package: bigmemory.sri
```

```
library(biglm)

library(biganalytics)
```

```
## Loading required package: foreach
```

```
zz <- biglm.big.matrix(y ~ x1 +x2+x3+x4+x5,data = d1)
summary(zz)
```

```
## Large data regression model: biglm(formula = formula, data = data, ...)
## Sample size =  10000000
##                       Coef           (95%           CI)         SE      p
## (Intercept) -2.499684e+08 -2.501058e+08 -2.498310e+08 68717.3599 0.0000
## x1           4.953410e+01  4.951760e+01  4.955050e+01     0.0082 0.0000
## x2           4.999437e+06  4.997611e+06  5.001263e+06   912.9757 0.0000
## x3          -3.618826e+04 -1.330310e+05  6.065448e+04 48421.3672 0.4548
## x4           1.104910e+01 -2.261006e+02  2.481987e+02   118.5748 0.9258
## x5          -8.167900e+00 -2.460320e+01  8.267400e+00     8.2176 0.3202
```

```
yy <- bigglm.big.matrix(y ~ x1 +x2+x3+x4+x5,data = d1)
sdd1<-summary(yy)$sigma
sd2<-sdd1*sqrt(10000000)
sd2
```

```
## numeric(0)
```

FF/BIG ANALYTICS

In bigmemory R keeps a pointer to a C++ matrix. The matrix is stored in RAM or on disk. In ff,R keeps metadata about the object, and the object is stored in a at binary file.

Advantages:

Allows R to work with multiple large Datasets. It also helps us to clean the system and not make a mess with tons of files. For modelling purposes, ffbase has bigglm.ffdf to allow to build generalized linear models easily on large data and can connect to the stream package for clustering & classification.

So if you had to make a choice between using bigmemory or ff, you can choose either as they both contribute similar performance.

```
library(ff)
```

```
## Loading required package: bit
## Attaching package bit
## package:bit (c) 2008-2012 Jens Oehlschlaegel (GPL-2)
## creators: bit bitwhich
## coercion: as.logical as.integer as.bit as.bitwhich which
## operator: ! & | xor != ==
## querying: print length any all min max range sum summary
## bit access: length<- [ [<- [[ [[<-
## for more help type ?bit
##
## Attaching package: 'bit'
## The following object is masked from 'package:base':
##
##     xor
## Attaching package ff
## - getOption("fftempdir")=="/var/folders/_g/4y3hwbzx15j_8b0nh3ql5sc00000gn/T//RtmpEXVsOT"
## - getOption("ffextension")=="ff"
## - getOption("ffdrop")==TRUE
## - getOption("fffinonexit")==TRUE
## - getOption("ffpagesize")==65536
## - getOption("ffcaching")=="mmnoflush"  -- consider "ffeachflush" if your system stalls on large write
## - getOption("ffbatchbytes")==16777216 -- consider a different value for tuning your system
## - getOption("ffmaxbytes")==536870912 -- consider a different value for tuning your system
##
## Attaching package: 'ff'
## The following objects are masked from 'package:bit':
##
##     clone, clone.default, clone.list
```

```
## The following object is masked from 'package:bigmemory':
##
##     is.readonly

## The following objects are masked from 'package:utils':
##
##     write.csv, write.csv2

## The following objects are masked from 'package:base':
##
##     is.factor, is.ordered
```

```
library(ffbase)
```

```
##
## Attaching package: 'ffbase'

## The following objects are masked from 'package:ff':
##
##     [.ff, [.ffdf, [<-.ff, [<-.ffdf

## The following objects are masked from 'package:base':
##
##     %in%, table
```

```
library(biglm)
library(biganalytics)

ddh <- bigglm.ffdf(y ~ x1 +x2+x3+x4+x5,data = d1)
sdd2<-summary(ddh)$sigma
sd3<-sdd2*sqrt(10000000)
sd3
```

```
## numeric(0)
```

RNetCDF

Is a package for reading and writing NetCDF Datasets. NetCDF is a widely used file format in atmospheric and oceanic research,especially for weather and climate model output,which allows storage of different types of array based data, along with a short data description.

DBI/RSQLite/sqldf/RPostgreSQL/RODBC

DBI: A database interface (DBI) definition for communication between R and relational database management systems. All classes in this package are virtual and need to be extended by the various R/DBMS implementations

RSQLite: This package embeds the SQLite database engine in R and provides an interface compliant with the DBI package. The source for the SQLite engine is included.

sqldf: This one is an outlier: Manipulate R data frames using SQL We cannot index a csv file or a data.frame. If you have to > repeatedly select subsets of your large data set, creating an index on the > relevant column in the sqlite table is an absolute life saver.

This is one reason the data.table package was created.It is very similar to a data.frame, with the addition of things like keys.If you consider an index in sqlite a life saver then data.table might be up your street.

Its really simple to do this in R.

It is extremely easy to use, and can be of great value to developers who need a database available but want to avoid the overhead often associated with installing and configuring an external database

```
set.seed(123);
n <- 5000;
p <- 5;
x <- matrix(rnorm(n * p), n, p);
x <- cbind(1, x);
bet <- c(2, rep(1, p));
y <- c(x %*% bet) + rnorm(n);

t1 <- Sys.time();
beta.hat <- solve(t(x) %*% x, t(x) %*% y);
t2 <- Sys.time();

gc();
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  10493022  560.4   32133844 1716.2  48394840 2584.6
## Vcells 274249460 2092.4  828725835 6322.7 828718762 6322.7
```

```
dat <- as.data.frame(x);

rm(x);
gc();
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  10493160  560.4   32133844 1716.2  48394840 2584.6
## Vcells 274249676 2092.4  828725835 6322.7 828718762 6322.7
```

```
dat$y <- y;
rm(y);
gc();
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  10493182  560.4   32133844 1716.2  48394840 2584.6
## Vcells 274249713 2092.4  828725835 6322.7 828718762 6322.7
```

```
colnames(dat) <- c(paste("x", 0:p, sep = ""), "y");
head(dat)
```

```
##   x0          x1         x2         x3         x4         x5         y
## 1  1 -0.56047565 -0.4941739  2.3707252 -1.3538489 -0.8362967 0.3307226
## 2  1 -0.23017749  1.1275935 -0.1668120 -0.5793773 -0.2205730 0.7955423
## 3  1  1.55870831 -1.1469495  0.9269614 -0.8610442 -2.1035148 0.9542624
## 4  1  0.07050839  1.4810186 -0.5681517  0.9726783 -1.6678075 2.8062886
## 5  1  0.12928774  0.9161912  0.2250901  0.6191458 -1.0979629 2.9994374
## 6  1  1.71506499  0.3351310  1.1319859  1.3854457 -1.6656212 4.4185995
```

```
gc();
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  10493276  560.5   32133844 1716.2  48394840 2584.6
## Vcells 274249943 2092.4  828725835 6322.7 828718762 6322.7
```

```
t3 <- Sys.time();
# Will also load the DBI package
```

```
library(RSQLite);
# Using the SQLite database driver
m <- dbDriver("SQLite");
# The name of the database file
dbfile <- "regression.db";
# Create a connection to the database
con <- dbConnect(m, dbname = dbfile);
# Write the data in R into database
if(dbExistsTable(con, "regdata")) dbRemoveTable(con, "regdata");
```

```
## [1] TRUE
```

```
dbWriteTable(con, "regdata", dat, row.names = FALSE);
```

```
## [1] TRUE
```

```
# Close the connection
dbDisconnect(con);
```

```
## [1] TRUE
```

```
# Garbage collection
t4 <- Sys.time();
rm(dat);
gc();
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  10523084  562.0   32133844 1716.2  48394840 2584.6
## Vcells 274237414 2092.3  828725835 6322.7 828718762 6322.7
```

```
t5 <- Sys.time();
m <- dbDriver("SQLite");
dbfile <- "~/Documents/NU/PM/Lab/Assignments/Regression2/regression.db";
con <- dbConnect(m, dbname = dbfile);
# Get variable names
vars <- dbListFields(con, "regdata");
xnames <- vars[-length(vars)];
yname <- vars[length(vars)];
# Generate SQL statements to compute X'X
mult <- outer(xnames, xnames, paste, sep = "*");
lower.index <- lower.tri(mult, TRUE);
mult.lower <- mult[lower.index];
sql <- paste("sum(", mult.lower, ")", sep = "", collapse = ",");
sql <- sprintf("select %s from regdata", sql);
txx.lower <- unlist(dbGetQuery(con, sql), use.names = FALSE);



txx <- matrix(0, p + 1, p + 1);
txx[lower.index] <- txx.lower;
txx <- t(txx);
txx[lower.index] <- txx.lower;
# Generate SQL statements to compute X'Y
sql <- paste(xnames, yname, sep = "*");
sql <- paste("sum(", sql, ")", sep = "", collapse = ",");
sql <- sprintf("select %s from regdata", sql);
```

```
txy <- unlist(dbGetQuery(con, sql), use.names = FALSE);

txy <- matrix(txy, p + 1);

# Compute beta hat in R
beta.hat.DB <- solve(txx, txy);
t6 <- Sys.time();
dbDisconnect(con);
```

```
## [1] TRUE
```

```
beta.hat
```

```
##           [,1]
## [1,] 1.9949264
## [2,] 1.0010715
## [3,] 1.0189432
## [4,] 1.0034469
## [5,] 0.9778601
## [6,] 1.0038719
```

```
beta.hat.DB
```

```
##           [,1]
## [1,] 1.9949264
## [2,] 1.0010715
## [3,] 1.0189432
## [4,] 1.0034469
## [5,] 0.9778601
## [6,] 1.0038719
```

```
max(abs(beta.hat - beta.hat.DB));
```

```
## [1] 0
```

```
t2 - t1;
```

```
## Time difference of 0.1426752 secs
```

```
t4 - t3;
```

```
## Time difference of 1.01684 secs
```

```
t6 - t5;
```

```
## Time difference of 0.04260588 secs
```

```
?t()
```

SNOW/SNOWFALL

Snow (Simple network of Workstations) provides an interface to several parallelization packages like MPI, PVM (Parallel Virtual Machines, etc.).All of these systems allow intrasystem communication for working with multiple CPUs, or intersystem communication for working with a cluster.

SnowFall: The Snowfall API uses list functions for parallelization. Calculations are distributed onto workers where each worker gets a portion of the full data to work with. The snowfall API is very similar to the snow API and has the following features:

Functions for loading packages and sources in the cluster Functions for exchanging variables between cluster nodes. All wrapper functions contain extended error handling. Changing cluster settings does not require changing R code. Can be done from command line. All functions work in sequential mode as well. Switching between modes requires no change in R code.

sfCluster

Generally in a cluster, all nodes have comparable performance specifications and each node will take approximately the same time to run.sfCluster wants to make sure that your cluster can meet your needs without exhausting resources.

The Advantages of using sfCluster

It checks your cluster to find machines with free resources if available. The available machines (or even the available parts of the machine) are built into a new sub-cluster which belongs to the new program. It can optionally monitor the cluster for usage and stop programs if they exceed their memory allotment, disk allotment, or if machines start to swap. It also provides diagnostic information about the current running clusters and free resources on the cluster including PIDs, memory use and runtime.

2.***Implicit Parallelism*** : Unlike explicit parallelism where the user controls (and can mess up) most of the cluster settings, with implicit parallelism most of the messy legwork in setting up the system and distributing data is avoided.

```
library(snowfall)
```

```
## Loading required package: snow
```

```
# 1. Initialisation of snowfall.
# (if used with sfCluster, just call sfInit())
sfInit(parallel=F, cpus=NULL,type = "SOCK")
```

```
## Warning in searchCommandline(parallel, cpus = cpus, type = type,
## socketHosts = socketHosts, : Unknown option on commandline:
## rmarkdown::render('/Users/Yash/Documents/NU/PM/Lab/Assignments/Regression2/
## Draft1f.Rmd',~+~~+~encoding~+~
```

```
## snowfall 1.84-6.1 initialized: sequential execution, one CPU.
```

```
# 2. Loading data.
require(mvna)
```

```
## Loading required package: mvna
```

```
data("sir.adm")
# 3. Wrapper, which can be parallelised.
wrapper <- function(idx) {
  # Output progress in worker logfile
  cat( "Current index: ", idx, "\n" )
  index <- sample(1:nrow(sir.adm), replace=TRUE)
  temp <- sir.adm[index, ]
  fit <- crr(temp$time, temp$status, temp$pneu)
  return(fit$coef)
}
# 4. Exporting needed data and loading required
# packages on workers.
sfExport("sir.adm")
```

```
## Warning in sfExport("sir.adm"): sfExport() writes to global environment in sequential mode.
```

```
sfLibrary(cmprsk)
```

```
## Loading required package: survival
```

```
# 5. Start network random number generator
# (as "sample" is using random numbers).
sfClusterSetupRNG()
```

```
## Warning in sfClusterSetupRNG(): Uniform random number streams (currently)
## not available in serial execution. Random numbers may differ in serial &
## parallel execution.
```

```
# 6. Distribute calculation


start <- Sys.time();
```

```
result <- sfLapply(1:1000, wrapper) ;
```

```
Sys.time()-start
```

```
## Time difference of 47.09631 secs
```

```
# Result is always in list form.
mean(unlist(result))
```

```
## [1] -0.9120328
```

```
# 7. Stop snowfall
sfStop()
```

```
# example process:
process <- function(parallel = FALSE,cpus = NULL){
  sfInit(parallel = parallel, cpus = cpus)
  sfLapply( 1:10^6, log10 )
  sfStop()
}

# computational time in sequential mode:

system.time(process(parallel = FALSE))
```

```
## Warning in searchCommandline(parallel, cpus = cpus, type = type,
## socketHosts = socketHosts, : Unknown option on commandline:
## rmarkdown::render('/Users/Yash/Documents/NU/PM/Lab/Assignments/Regression2/
## Draft1f.Rmd',~+~~+~encoding~+~
```

```
## snowfall 1.84-6.1 initialized: sequential execution, one CPU.
```

```
##    user  system elapsed
##   0.671   0.023   0.696
```

```
system.time(process(parallel =TRUE,cpus=2))
```

```
## Warning in searchCommandline(parallel, cpus = cpus, type = type,
## socketHosts = socketHosts, : Unknown option on commandline:
## rmarkdown::render('/Users/Yash/Documents/NU/PM/Lab/Assignments/Regression2/
## Draft1f.Rmd',~+~~+~encoding~+~
```

```
## snowfall 1.84-6.1 initialized (using snow 0.4-2): parallel execution on 2 CPUs.
```

```
##
## Stopping cluster

##    user  system elapsed
##   0.464   0.071   2.847
```

HadoopStreaming

MapReduce is a way of dividing a large job into many smaller jobs producing an output, and then combining the individual outputs into one output. It is a classic divide and conquer approach that is parallel and can easily be distributed among many cores or a CPU, or among many CPUs and nodes.

mapReduce is a pure R implementation of MapReduce. Many authors state that mapReduce is simply:

apply(map(data), reduce)

By default, mapReduce uses the same parallelization functionality as sapply.

For Example: mapReduce (map, ., data, apply = sapply)

In Simple terms:

Map: Performs operations such as filtering and sorting of the data ***Reduce***: Performs Summary operations on the mapped data

Hadoop: Hadoop is an open-source implementation of MapReduce that has gained a lot of traction in the data mining community. Hadoop is distributed with Hadoop Streaming, which allows map/reduce jobs to be written in any language including R.

The hadoopstreaming package is used in R to run map/reduce.

Rhipe

Rhipe is another R interface for Hadoop that provides following advantages : Incorporates an rhlapply function similar to the standard apply variants. Uses Google Protocol Buffers. Great flexibility in modifying Hadoop parameters. At the end of the day, which package in R will you choose to handle Big Data?

For datasets with size in the range 10GB, bigmemory and ff handle themselves well. For Datasets in the range of TB and PB, you can interface Hadoop in R

RHadoop

RHadoop is a collection of five R packages that allow users to manage and analyze data with Hadoop. The packages have been tested (and always before a release) on recent releases of the Cloudera and Hortonworks Hadoop distributions and should have broad compatibility with open source Hadoop and mapR's distribution. We normally test on recent Revolution R/Microsoft R and CentOS releases, but we expect all the RHadoop packages to work on a recent release of open source R and Linux.

rhdfs This package provides basic connectivity to the Hadoop Distributed File System. R programmers can browse, read, write, and modify files stored in HDFS from within R. Install this package only on the node that will run the R client.

rhbase This package provides basic connectivity to the HBASE distributed database, using the Thrift server. R programmers can browse, read, write, and modify tables stored in HBASE from within R. Install this package only on the node that will run the R client.

plyrmr This package enables the R user to perform common data manipulation operations, as found in popular packages such as plyr and reshape2, on very large data sets stored on Hadoop. Like rmr, it relies on Hadoop MapReduce to perform its tasks, but it provides a familiar plyr-like interface while hiding many of the MapReduce details. Install this package only every node in the cluster. rmr2 A package that allows R developer to perform statistical analysis in R via Hadoop MapReduce functionality on a Hadoop cluster. Install this package on every node in the cluster.

ravro A package that adds the ability to read and write avro files from local and HDFS file system and adds an avro input format for rmr2. Install this package only on the node that will run the R client.