

Big Data Analysis with Revolution R Enterprise

By Joseph Rickert
January 2011

Background

The R language is well established as the language for doing statistics, data analysis, data-mining algorithm development, stock trading, credit risk scoring, market basket analysis and all manner of predictive analytics. However, given the deluge of data that must be processed and analyzed today, many organizations have been reticent about deploying R beyond research into production applications.

The main barrier is that R is a memory-bound language. All data used in calculations — vectors, matrices, lists, data frames, and so forth — all need to be held in memory. Even for modern computers with 64-bit address spaces and huge amounts of RAM, dealing with data sets that are tens of gigabytes and hundreds of millions of rows (or larger) can present a significant challenge. The problem isn't just one of capacity, that is, being simply being able to accommodate the data in memory for analysis. For mission-critical applications, performance is also a prime consideration: if the overnight analysis does not complete in time for the open of business the next day, that's just as much of a failure as an out-of-memory error. And with data set sizes growing rapidly, scalability is also of concern: even if the in-memory analysis completes today, the IT manager still needs the confidence that the production run will complete — on time! — as the data set grows.

Revolution Analytics has addressed these capacity, performance and scalability challenges with its "Big Data" initiative to extend the reach of R into the realm of production data analysis with terabyte-class data sets. This paper describes Revolution Analytics' new add-on package called *RevoScaleR*[™], which provides unprecedented levels of performance and capacity for statistical analysis in the R environment. For the first time, R users can process, visualize and model their largest data sets in a fraction of the time of legacy systems, without the need to deploy expensive or specialized hardware.

Limitations of In-memory Data Analysis

As mentioned above, R requires all data to be loaded into memory for processing. This design feature limits the size of files that can be analyzed on a modest desktop computer. For example, in the following line of code, the data frame, `myData`, contains 5,000,000 rows and three columns, only a very small subset of the airline data file that will be examined later in this paper. The error message from the R interpreter shows that even this relatively small file cannot be processed in the R environment on a PC with 3 gigabytes of memory.

```
> lm(ArrDelay~UniqueCarrier+DayOfWeek, data=myData)
```

```
Error: cannot allocate vector of size 751.8 Mb
```

Of course, a high-end PC or server configured with much more memory and running a 64-bit version of R can go a considerable way in extending the memory barrier. But even when the data can fit into memory, the performance of standard R on large files can be prohibitively slow. Really breaking through the memory/performance barrier requires implementing external memory algorithms and data structures that explicitly manage “data placement and movement”.^[2] *RevoScaleR* brings parallel external memory algorithms and a new very efficient data file format to R.

RevoScaleR Features

The *RevoScaleR* package provides a mechanism for scaling the R language to handle very large data sets. There are three major components to this package:

- A new file format especially designed for large files,
- External memory implementations of the statistical algorithms most commonly used with large data sets, and
- An extensible programming framework that allows R and later C++ programmers to write their own external memory algorithms that can take advantage of Revolution R Enterprise’s new Big Data capabilities.

RevoScaleR XDF File Format

RevoScaleR provides a new data file type with extension `.xdf` that has been optimized for “data chunking”, accessing parts of an Xdf file for independent processing. Xdf files store data in a binary format. Methods for accessing these files may use either horizontal (rows) or vertical (columns) block partitioning. The file format provides very fast access to a specified set of rows for a specified set of columns. New rows and columns can be added to the file without re-writing the entire file. *RevoScaleR* also provides a new R class, `RxDataSource`, that has been designed to support the use of external memory algorithms with `.xdf` files.

Table 1 lists the key functions for working with Xdf files:

Table 1-Key Xdf Functions

Function	Description
rxTextToXdf	Import a text data file into an Xdf file
rxDataFrameToXdf	Write a data frame in memory to an Xdf file
rxDataStepXdf	Transform data from an input Xdf file to and output .xdf file
rxImportToXdf	Import SAS file or fixed-format text file
rxReadXdf	Read data from an Xdf file into a data frame
rxXdfToText	Export contents of Xdf file to csv file
rxGetInfoXdf	Get information about an Xdf file
rxGetVarInfoXdf	Get the variable information for an Xdf file: column names, descriptions, factor labels etc.
rxSetVarInfoXdf	Modify the variable information for an Xdf file, including names, descriptions and factor labels
rxDataStepXdf	Create a new Xdf file with variable selections and transformations

Table 2 lists some of the available low level functions for working with chunks of data. The *RevoScaleR: Getting Started Guide* presents detailed examples for working with these functions.^[1]

Table 2 - RxDataSource Functions

Function	Description
rxOpenData	Opens the data file for reading and returns a handle
rxReadNext	Reads the next chunk of data from the specified file using the data handle
rxReadAll	Reads all of the data from the file specified by the data handle and creates a data frame
rxCloseData	Closes the data file specified by the data handle

Statistical Algorithms

The *RevoScaleR* package also contains parallel external memory implementations of the most common algorithms used to analyze very large data files. These functions, which are listed in Table 3, represent the initial release of the *RevoScaleR* package. Additional algorithms are planned for subsequent releases.

Table 3 - Statistical Functions in *RevoScaleR*

Function	Description
rxSummary	Computes summary statistics on data in an Xdf file or in a data frame
rxCrossTabs	Creates contingency tables by cross-classifying factors in an Xdf file or data frame. This function permits multi-dimensional cross classification
rxCube	Create a list or data frame of cross tabulations in long format
rxLinMod	Fits linear models from data residing in an Xdf file or data frame. rxLinModels fits both simple and multiple regression models.
rxLogit	Fits binomial, logistic regression models from data residing in and Xdf file or data frame
rxPredict	Computes predicted values and residuals for the models created by rxLinMod and rxLogit

All of the *RevoScaleR* statistical functions produce objects that may be used as input to standard R functions.

Extensible Programming Framework

Advanced R users can write their own functions to exploit the capabilities of the Xdf files and RxdataSource objects. Any statistical or data mining function that can work on chunks of data and be implemented as an external memory algorithm is a candidate for a *RevoScaleR* implementation. This R programming interface is available in the current release of *RevoScaleR*; a subsequent release will add a C++ programming interface.

***RevoScaleR* Examples**

This section illustrates many of the capabilities of the *RevoScaleR* package through an extended example that uses the moderately large airlines data file: AirlineData87to08. This file contains

flight arrival and departure details for all commercial flights within the United States for the period October 1987 through April 2008. The file contains 123,534,969 rows and 29 columns (variables). It requires 13,280,963 KB (over 13 Gb) to store, uncompressed on disk.

Code Block 1 uses the rxOptions function to set the environment. For a more robust computer, the number of cores can also be set appropriately using the rxOptions function. (For the Intel Xeon dual 6-core processor server used in the benchmarks below, setting this value to 12, the number of physical cores, produced optimal results.)

Code Block 1

```
rxOptions(sampleDataDir = system.file("SampleData", package="RevoScaleR"),
          demoScriptsDir = system.file("demoScripts", package="RevoScaleR"),
          blocksPerRead = 1,
          reportProgress = 2,
          rowDisplayMax = -1,
          memStatsReset = 0,
          memStatsDiff = 0,
          numCoresToUse = 2,
          showTransformFn = FALSE)
```

Code Block 2 access the file stored on disk in the Xdf format, displays information about the file and its variables (see Table 4) and produces a summary (Table 5) for two variables: Arrdelay and DayofWeek.

Code Block 2

```
defaultDataDir <- "C:/Users/ . . ./revoAnalytics"
dataName <- file.path(defaultDataDir, "AirlineData87to08")
rxGetInfoXdf(dataName, getVarInfo=TRUE)
# Get summary data
rxSummary(~ArrDelay:DayOfWeek, data=dataName)
```

Table 4 – Information and Variables in Airline File

```
> rxGetInfoXdf(dataName, getVarInfo=TRUE)
```

Name:	C:\Users\ . . \AirlineData87to08.xdf
Number of rows:	123534969
Number of variables:	29
Number of blocks:	832

Variable Information:

Var 1: Year, Type: factor
22 factor levels: 1987 1988 1989 1990 1991 ... 2004 2005 2006 2007 2008
Var 2: Month, Type: factor
12 factor levels: January February March April May ... August September October November December
Var 3: DayofMonth, Type: factor
31 factor levels: 1 2 3 4 5 ... 27 28 29 30 31
Var 4: DayOfWeek, Type: factor
7 factor levels: Monday Tuesday Wednesday Thursday Friday Saturday Sunday
Var 5: DepTime, Type: numeric, Storage: float32, Low/High: (0.0167, 29.5000)
Var 6: CRSDepTime, Type: numeric, Storage: float32, Low/High: (0.0000, 24.0000)

Var 7: ArrTime, Type: numeric, Storage: float32, Low/High: (0.0167, 29.9167)
Var 8: CRSArrTime, Type: numeric, Storage: float32, Low/High: (0.0000, 24.0000)
Var 9: UniqueCarrier, Type: factor
29 factor levels: 9E AA AQ AS B6 ... UA US WN XE YV
Var 10: FlightNum, Type: factor
8160 factor levels: 1451 1453 1454 1455 1457 ... 9742 9743 6813 6913 6513
Var 11: TailNum, Type: factor
13537 factor levels: NA N7298U N7449U N7453U N7288U ... N516AS N763JB N766JB
N75428 N75429
Var 12: ActualElapsedTime, Type: integer, Low/High: (-719, 1883)
Var 13: CRSElapsedTime, Type: integer, Low/High: (-1240, 1613)
Var 14: AirTime, Type: integer, Low/High: (-3818, 3508)
Var 15: ArrDelay, Type: integer, Low/High: (-1437, 2598)
Var 16: DepDelay, Type: integer, Low/High: (-1410, 2601)
Var 17: Origin, Type: factor
347 factor levels: SAN SFO BUR OAK LAX ... ROW GCC RKS MKG OTH
Var 18: Dest, Type: factor
352 factor levels: SFO RNO OAK BUR LAX ... PIR GCC RKS MKG OTH
Var 19: Distance, Type: integer, Low/High: (0, 4983)
Var 20: TaxiIn, Type: integer, Low/High: (0, 1523)
Var 21: TaxiOut, Type: integer, Low/High: (0, 3905)
Var 22: Cancelled, Type: logical, Storage: uchar, Low/High: (0, 1)
Var 23: CancellationCode, Type: factor
5 factor levels: NA carrier weather NAS security
Var 24: Diverted, Type: logical, Storage: uchar, Low/High: (0, 1)
Var 25: CarrierDelay, Type: integer, Low/High: (0, 2580)
Var 26: WeatherDelay, Type: integer, Low/High: (0, 1510)
Var 27: NASDelay, Type: integer, Low/High: (-60, 1392)
Var 28: SecurityDelay, Type: integer, Low/High: (0, 533)
Var 29: LateAircraftDelay, Type: integer, Low/High: (0, 1407)

Table 5 – Summary of variables ArrDelay and DayofWeek

Summary Statistics for: ArrDelay:DayOfWeek (Total: 123534969, Missing: 2587529)					
Name	Mean	StdDev	Min	Max	ValidObs
ArrDelay:DayOfWeek	7.049963	30.75081	-1437	2598	120947440
Statistics by category (7 categories):					
Category	Means	StdDev	Min	Max	ValidObs
ArrDelay for DayOfWeek=Monday	6.669515	30.17812	-1410	1879	17750849
ArrDelay for DayOfWeek=Tuesday	5.960421	29.06076	-1426	2137	17643973
ArrDelay for DayOfWeek=Wednesday	7.091502	30.37856	-1405	2598	17697936
ArrDelay for DayOfWeek=Thursday	8.945047	32.30101	-1395	2453	17683723
ArrDelay for DayOfWeek=Friday	9.606953	33.07271	-1437	1808	17707329
ArrDelay for DayOfWeek=Saturday	4.187419	28.29972	-1280	1942	15617054
ArrDelay for DayOfWeek=Sunday	6.52504	31.11353	-1295	2461	16846576

Next we perform a simple linear regression of ArrDelay against DayOfWeek using the “cube” option of rxLinMod, the *RevoScaleR* function to fit linear models. When cube is set to TRUE and the first explanatory variable in the regression model is categorical, rxLinMod uses a partitioned inverse algorithm to fit the model. This algorithm can be faster and uses less memory than the default inverse algorithm. Code Block 3 shows the code for fitting the model and uses the standard R functions to obtain the results of the regression (Table 6) and plot Arrival delay by day of week (Figure 1).

Note that the p-values calculated by the standard t-test are extremely small, as would be expected by using such a simple model with such a large data set. Also note that the “cube” option produces a data frame as output that shows the counts of the data points that went into producing each coefficient (Table 7).

Code Block 3

```
# Fit a linear model with the cube option
arrDelayLml <- rxLinMod(ArrDelay ~ -1+DayOfWeek, data=dataName, cube=TRUE)
summary(arrDelayLml) # Use the standard R function summary
arrDelayLml$countDF # Look at the output from the cube option
# Plot arrival delay by day of week
xyplot( ArrDelay ~ DayOfWeek, data = arrDelayLml$countDF, type = "l",
        lwd=3,pch=c(16,17), auto.key=TRUE)
```

Table 6 – Output of summary

rxLinMod.formula(formula = ArrDelay ~ -1 + DayOfWeek, data = dataName, cube=TRUE)				
Coefficients:	Estimate	Std. Error	t value	Pr(> t)
DayOfWeek.Monday	6.670	0.007288	915.1	2.22E-16
DayOfWeek.Tuesday	5.960	0.007310	815.4	2.22E-16
DayOfWeek.Wednesday	7.092	0.007299	971.6	2.22E-16
DayOfWeek.Thursday	8.945	0.007302	1225	2.22E-16
DayOfWeek.Friday	9.607	0.007297	1316.6	2.22E-16
DayOfWeek.Saturday	4.187	0.007770	538.9	2.22E-16
DayOfWeek.Sunday	6.525	0.007481	872.2	2.22E-16
Residual standard error: 30.71 on 120947433 degrees of freedom				
Multiple R-squared: 0.002933, Adjusted R-squared:0.002933				
F-statistic: 5.93e+04 on 6 and 120947433 DF, p-value: < 2.2e-16				

Figure 1 – Average Arrival Delay by Day of Week

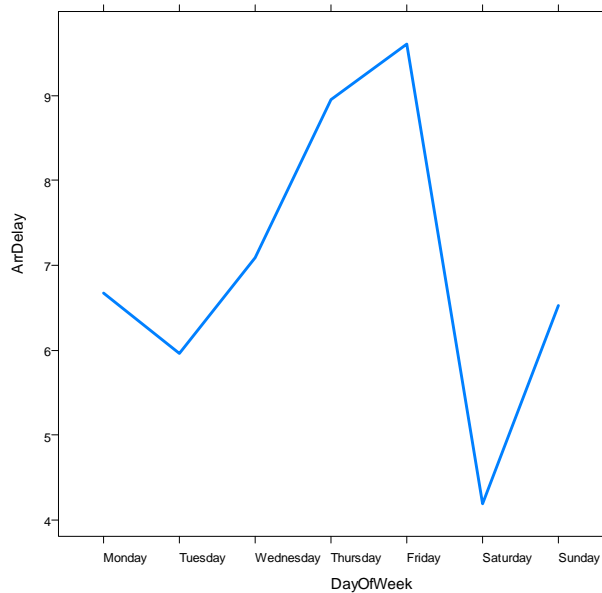


Table 7 – Output from cube option

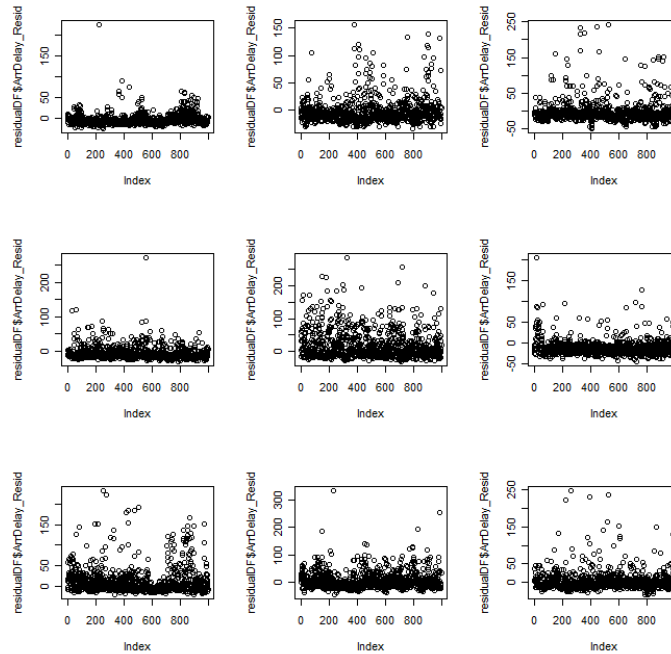
arrDelayLm1\$countDF		
DayOfWeek	ArrDelay	Counts
Monday	6.669515	17750849
Tuesday	5.960421	17643973
Wednesday	7.091502	17697936
Thursday	8.945047	17683723
Friday	9.606953	17707329
Saturday	4.187419	15617054
Sunday	6.52504	16846576

The *RevoScaleR* package also contains a predict function for predicting values of a linear model and computing residuals. The rxPredict function in Code Block4 appends the variables ArrDelay_Pred and ArrDelay_Resid as columns 30 and 31, respectively, to the airlines data file. Figure 2 which shows 9 residual plots of 1000 residuals randomly selected from the file Airlinedata87To08, illustrates how the residuals are available to R for further analysis.

Code Block 4

```
# Predict function computes predicted values and residuals
rxPredict(modelObject=arrDelayLm1,data=dataName,computeResiduals=TRUE)
par(mfrow=c(3,3))
start <- runif(16,1,12000000)
for (i in 1:9){
  residualDF <- rxReadXdf(file=dataName,varsToKeep="ArrDelay_Resid",
    startRow= start[i],numRows=1000)
  plot(residualDF$ArrDelay_Resid)}
```

Figure 2 – 1000 residuals from 4 random locations*



* 31730477 72004173 83622653 6801174 47990749 99815488 17721542 88637507
43808465

Next, we consider a multiple regression of arrival delay (ArrDelay) on the day of the week (DayOfWeek) and departure time (CRSDepTime). The second line of Code Block 5 constructs the linear model to carry out the regression and illustrates the “inline” use of the “F” function which makes a factor variable out of CRSDepTime on the fly as it is being used to construct the linear model. This function illustrates a fundamental design concept of *RevoScaleR*: the ability efficiently transform data, and create new variables without having to make multiple passes through the file. The cube option produces a table of arrival delay counts by departure time and day of the week. The first five lines of this output are displayed in Table 8. Figure 3 show a plot of arrival delay by departure time and day of the week that is based on these data.

Code Block 5

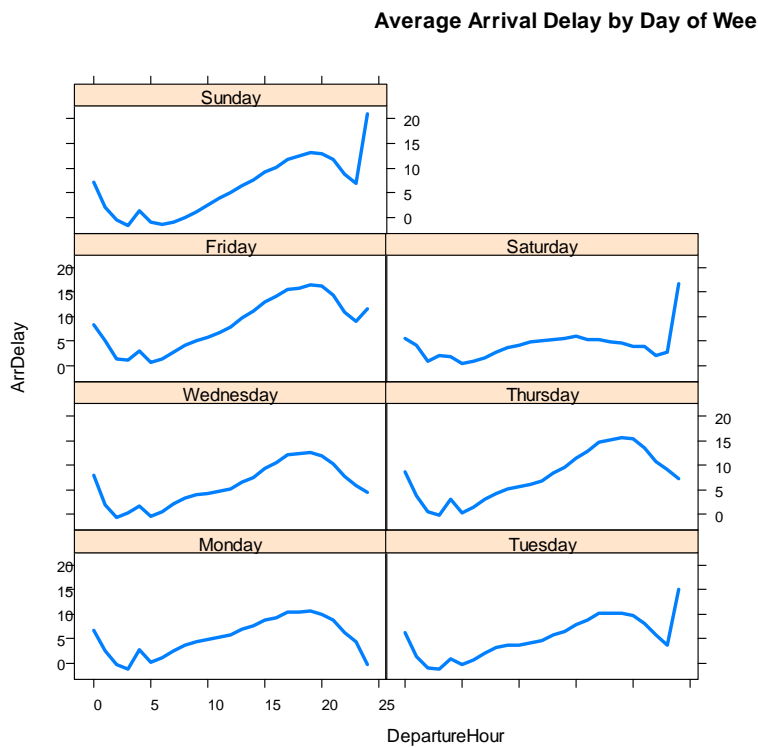
```
# Multiple Regression
arrDelayLm2 <- rxLinMod(ArrDelay ~ DayOfWeek:F(CRSDepTime), data=dataName,cube=TRUE)
arrDelayDT <- arrDelayLm2$countDF
arrDelayDT[1:5,]
summary(arrDelayLm2)
# Plot data
names(arrDelayDT) <- c("DayOfWeek", "DepartureHour", "ArrDelay", "Counts")
arrDelayDT$DepartureHour <- as.numeric(as.character(arrDelayDT$DepartureHour))
xyplot(ArrDelay ~ DepartureHour|DayOfWeek, data = arrDelayDT,
type = "l", lwd=3,pch=c(16,17),
main='Average Arrival Delay by Day of Week by Departure Hour', layout=c(2,4),
auto.key=TRUE)
```

Table 8. Output from Cube Option

```
> arrDelayDT[1:5,]
```

Row	DayOfWeek	X.CRSDepTime	ArrDelay Counts
1	Monday	0 6.687033	142344
2	Tuesday	0 6.318279	129261
3	Wednesday	0 7.891790	128777
4	Thursday	0 8.692393	125657
5	Friday	0 8.381638	126683

Figure 3. Plot of Data produced by Cube Option



Finally, we illustrate the flexible way the *RevoScaleR* function `rxDataStepXdf` can be used to generate a file containing a subset of the airlines data and new variables that are transformations of some of the variables in the airlines data file. These variables are created “on the fly” with the help of the `transformFunc` parameter to the `rxDataStepXdf` function. The first six lines of Code Block 6 are a function that defines the new variable. Lines 9 and 10 show the `rxDataStepXdf` function which reads the airline data file, creates an new file keeping only the variables designated and transforms these variables using the specified transformation function.

Code Block 6

```
# Create function to transform data
myTransforms <- function(data){
data$Late <- data$ArrDelay > 15
data$DepHour <- as.integer(data$CRSDepTime)
data$Night <- data$DepHour >= 20 | data$DepHour <= 5
return(data)}
# The rxDataStepXdf function read the existing data set, performs the
# transformations, and creates a new data set.
rxDataStepXdf(inFile = dataName, outFile="ADS2", transformFunc=myTransforms,
varsToKeep=c("ArrDelay","CRSDepTime","DepTime"))
#
rxGetInfoXdf(file="ADS2",getVarInfo=TRUE)
# Run a logistic regression using the new variables
logitObj <- rxLogit(Late~DepHour+Night, data="ADS2", verbose=TRUE)
```

Table 9 shows the meta-data and data for the newly created file, called ADS2. Table 10 shows the output from performing a logistic regression using the newly defined variables.

Table 9. Description of data file ADS2

```
> rxGetInfoXdf(file="ADS2",getVarInfo=TRUE)
File name: C:\. . . \ADS2.xdf
Number of rows: 123534969
Number of variables: 6
Number of blocks: 832
```

Variable information:

```
Var 1: ArrDelay, Type: integer, Low/High: (-1437, 2598)
Var 2: CRSDepTime, Type: numeric, Storage: float32, Low/High: (0.0000, 24.0000)
Var 3: DepTime, Type: numeric, Storage: float32, Low/High: (0.0167, 29.5000)
Var 4: Late, Type: logical, Storage: uchar, Low/High: (0, 1)
Var 5: DepHour, Type: integer, Low/High: (0, 24)
Var 6: Night, Type: logical, Storage: uchar, Low/High: (0, 1)
```

Table 10. Output from logistic regression

```
Logistic Regression Results for: Late ~ DepHour + Night
Dependent Variable: Late
Total independent variables: 3
Number of valid observations: 120947440 (Number excluded for missings: 2587529)
-2*LogLikelihood: 1.17048e+008 (Residual Deviance on 120947437 degrees of freedom)
Condition number of final VC matrix: 67.1834
```

Row	Coeffs.	Value	Std. Error	t Value	Pr(> t)
[1,]	(Intercept)	-2.3573	0.0008	-2960.3166	0.0000
[2,]	DepHour	0.0705	0.0001	1219.4029	0.0000
[3,]	Night	-0.2033	0.0008	-255.8746	0.0000

RevoScaleR Benchmarks

In order to provide some idea of the performance that can be reasonably expected from *RevoScaleR* functions operating on moderately sized data sets, this section provides benchmarks of *rxLinMod*, *rxCrossTabs* and *rxLogit*. All benchmarks were conducted on two platforms: (1) a Lenovo Thinkpad laptop with dual-core, Intel P8600 2.40GHz processor and 3 GB of RAM running Windows 7, and (2) an Intel Xeon X5660 server with 2 2.80 GHz CPUs each with 6 cores and 12 threads with 72GB of RAM running Windows Server 2008. While no formal comparisons are against other well known statistical systems are included in this analysis, we believe that *RevoScaleR*'s capacity and performance abilities substantially out-perform competitive products.

rxLinMod

The file *AirlineData87to08* contains information on flight arrivals and departure details for all commercial flights within the US from October 1987 to April. It is 13,280,936 KB and contains 123,534,969 rows and 29 columns. Table 4 above provides a summary of the information in the file. The following R code (Code Block 7) runs a simple regression followed by a multiple regression with three explanatory variables. Note that the explanatory variables are categorical. The *cube* option for *rxLinMod* ensures the efficient processing of categorical data.

Code Block 7

```
#rxOptions(numCoresToUse=12)
defaultDataDir <- "C:/Users/. . ./revoAnalytics"
dataName <- file.path(defaultDataDir,"AirlineData87to08")
rxGetInfoXdf(dataName,getVarInfo=TRUE)
# Simple Regression
system.time(delayArr <- rxLinMod(ArrDelay ~
DayOfWeek,data=dataName,blocksPerRead=30))
# Multiple Regression
system.time(delayCarrierLoc <- rxLinMod(ArrDelay ~
UniqueCarrier+Origin+Dest,data=dataName,blocksPerRead=30,cube=TRUE))
```

Table 11 contains the results of the regression benchmarks. Note that the average time for first runs of the simple regression on the laptop is considerably longer than the average time to complete subsequent runs of the same regression. This was most likely due to disk caching on the laptop. This phenomenon was not observed when doing the multiple regression on the laptop and it was not observed at all on the server. Performance on the server was consistent among multiple runs. Other than setting the number of cores to be used by the *RevoScaleR* compute engine equal to 12, the number of real cores available on the server, no attempt was made to optimize the performance of the server.

Table 11 – Regression Benchmark Results

	Average Elapsed Time (seconds)	
	Laptop	Server
Simple Regression		
first run	38.74	NA
subsequent runs	4.05	2.6
Multiple Regression	85.9	21.01

rxCrossTabs

The file Censuslp2001 contains US census data. It is 17,079,992 KB, and contains 14,583,271 rows and 265 columns. Table 12 contains a portion of the output describing the file Censuslp2001 that is produced by the command rxGetInfoXdf.

Table 12 - Header Information and first five columns of Censuslp2001

```

> rxGetInfoXdf(dataFile,getVarInfo=TRUE)
File name: C:\. . . \Censuslp2001.xdf
Number of rows: 14583731
Number of variables: 265
Number of blocks: 487
Variable information:
Var 1: rectype
      2 factor levels: H P
Var 2: year, Census year
      14 factor levels: 2000 1850 1860 1870 1880 ... 1950 1960 1970 1980 1990
Var 3: datanum, Data set number
      Type: integer, Low/High: (1, 1)
Var 4: serial, Type: integer, Low/High: (1, 6175965)
Var 5: numprec, Number of person records following
      Type: integer
      55 factor levels: 0 1 2 3 4 ... 50 51 52 53 54
    
```

The following R code (Code Block 8) runs the benchmark. Note that the text string in the third line must include the entire path to the directory containing the file

Code Block 8

```

#rxOptions(numCoresToUse=12)      # Only run on the Server
defaultDataDir <- "C:/Users/. . . "
dataFile <- file.path(defaultDataDir,"Census5PCT2000")
# Use rxCrossTabs to compute a 2-way cross tabulation:
system.time(ageSex <- rxCrossTabs(~F(age):sex,data=dataFile,
                                  pweights="perwt",blocksPerRead=25))
# Compute a 4-way cross tabulation
system.time(asmc <- rxCrossTabs(~F(age):sex:F(marst):F(condo),
                                data=dataFile,pweights="perwt",blocksPerRead=25))
    
```

Table 13 presents the results of the benchmarks. Notice that the first time the 2-way cross-tab was run on the laptop it took an average of 14.17 seconds to complete, while subsequent runs completed in little over a second. This phenomenon is most likely due to disk caching. Also note that the 4-way cross-tab on the laptop ran slightly faster, on average, than the 2-way cross-tab. This effect is also most likely due to disk caching as the laptop was not rebooted between runs.

Table 13 - Results of rxCrossTabs Benchmark

	Average Elapsed Time (seconds)	
	Laptop	Server
2-way cross tab		
first run	14.17	NA
subsequent runs	1.2	1.78
4-way cross tab		
first run	11.26	NA

subsequent runs	1.56	2.9
-----------------	------	-----

Performance on the server was consistent among multiple runs. No first run effect was observed which is to be expected. Other than setting the number of cores to be used by the *RevoScaleR* compute engine equal to 12, the number of real cores available on the server, no attempt was made to optimize the performance of the server.

rxLogit

The file `mortDefault` contains ten years of mortgage default data (2000 to 2009). It is 234,378 KB, has 10,000,000 and 6 variables (Table 14). Code Block 9 presents function to run the regression and Table 15 contains the benchmark results.

Table 14 – Header Information and Variables in Mortgage File

```

> rxGetInfoXdf(dataFileName,getVarInfo=TRUE)
File name: C:\. . \mortDefault.xdf
Number of rows: 10000000
Number of variables: 6
Number of blocks: 10
Variable information:
Var 1: creditScore, Type: integer, Low/High: (432, 955)
Var 2: houseAge
      41 factor levels: 0 1 2 3 4 ... 36 37 38 39 40
Var 3: yearsEmploy, Type: integer, Low/High: (0, 15)
Var 4: ccDebt, Type: integer, Low/High: (0, 15566)
Var 5: year
      10 factor levels: 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009
Var 6: default, Type: integer, Low/High: (0, 1)
    
```

Code Block 9

```

# Run a logistic regression on the file
system.time(logitObj <- rxLogit(default~creditScore + yearsEmploy + ccDebt
                               + houseAge + year,data=dataFileName,
                               blocksPerRead=2, verbose=1,reportProgress = 1))
    
```

Table 15 – Results of rxLogit Benchmarks

	Average Elapsed Time (seconds)	
	Laptop	Server
Logistic Regression	65.37	22.17

Summary

The *RevoScaleR* package from Revolution Analytics provides external memory algorithms that help R break through the memory/performance barrier. The XDF file format provides an efficient and flexible mechanism for processing large data sets. The new package provides functions for creating XDF files from text files, writing XDF data to text files and data frames and for transforming and manipulating variables during the read/ write process.

The new package also contains statistical functions for generating summary statistics, performing multi-way cross tabulations on large data sets and for developing linear models and performing logistic regressions. Preliminary benchmarks show that *RevoScaleR* functions are fast and efficient -- enabling real data analysis to be performed on a 120 + million row, 13GB data set on a common dual core laptop. Moreover, the benchmarks show that the performance of *RevoScaleR* functions scale nicely as computational resources increase.

References

[1] Revolution Analytics. *RevoScaleR: Getting Started Guide*, July 19, 2010

[2] Vitter, Jeffrey Scott. *Algorithms and Data Structures for External Memory*. Now Publishers, Inc.: Hannover, MA 2008

About Revolution Analytics

Revolution Analytics delivers advanced analytics software at half the cost of existing solutions. Led by predictive analytics pioneer and SPSS co-founder Norman Nie, the company brings high performance, productivity, and enterprise readiness to open source R, the most powerful statistics language in the world.

In the last 10 years, R has exploded in popularity and functionality and has emerged as the data scientists' tool of choice. Today R is used by over 2 million analysts worldwide in academia and at cutting-edge analytics-driven companies such as Google, Facebook, and LinkedIn. To equip R for the demands and requirements of all business environments, Revolution R Enterprise builds on open source R with innovations in big data analysis, integration and user experience.

The company's flagship Revolution R product is available both as a workstation and server-based offering. Revolution R Enterprise Server is designed to scale and meet the mission-critical production needs of large organizations such as Merck, Bank of America and Mu Sigma, while Revolution R Workstation offers productivity and development tools for individuals and small teams that need to build applications and analyze data.

Revolution Analytics is committed to fostering the growth of the R community. The company sponsors the Inside-R.org community site, local users groups worldwide, and offers free licenses of Revolution R Enterprise to everyone in academia to broaden adoption by the next generation of data scientists. Revolution Analytics is headquartered in Palo Alto, Calif. and backed by North Bridge Venture Partners and Intel Capital.

Please visit us at www.revolutionanalytics.com