D04

巨量資料於
R軟體中的處理與分析

(High-Performance Computing with R)

吳漢銘

國立政治大學 統計學系

https://hmwu.idv.tw

Outline

The Definition of Big Data is a Moving Target

http://www.opentracker.net/article/definitions-big-data

opentracker
event tracking and analytics

PRODUCTS FEATURES PRICI

Start your free, **no-risk**, 4 we

Definitions of Big Data

Q: Can you please provide me with a definition of B

A: The definition of Big Data is a moving target.

In order to make it possible to follow the discussion, as it evolves, we see have started a list of definitions of Big Data, as we read them on the internet.

Author names: Andrew Brust (ZDNet), Bill Franks (FCW article),

PCm
Weat
Mike
(O'Re
Libra
Rous
(O'Re
Slash
O'Rei
Evels
Stepl

31. "Big data is a popular term used to describe the exponen information, both structured and unstructured. Ultimately, regardless of the factors involved, we believe applies (per Gartner's assessment) whenever an organiz data exceeds its current capacity." SAS.

32. The simplest definition of "Big Data" is "it doesn't fit in Ex from the full quote; "I have joked that the simplest defin - and when you think of it, it's true for most people who traditional approach to a Big Data one."
Stephane Hamel comment 8/2012 Big Data - What It Me

33. More to follow...

updated: April 26, 2013

Application Delivery Strategies — META Group

Date: 6 February 2001

File: 949
Author: Doug Laney

3D Data Management: Controlling Data Volume, Velocity, and Variety. Current business conditions and mediums are pushing traditional data management principles to their limits, giving rise to novel, more formalized approaches.

Report | McKinsey Global Institute

Big data: The next frontier for innovation, competition, and productivity

May 2011 | by James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, Angela Hung Byers

Download » Executive Summary PDF–922KB » Full Report PDF–6MB » Kindle MOBI–4MB » eBook EPUB–3MB

PDF Print E-mail Share

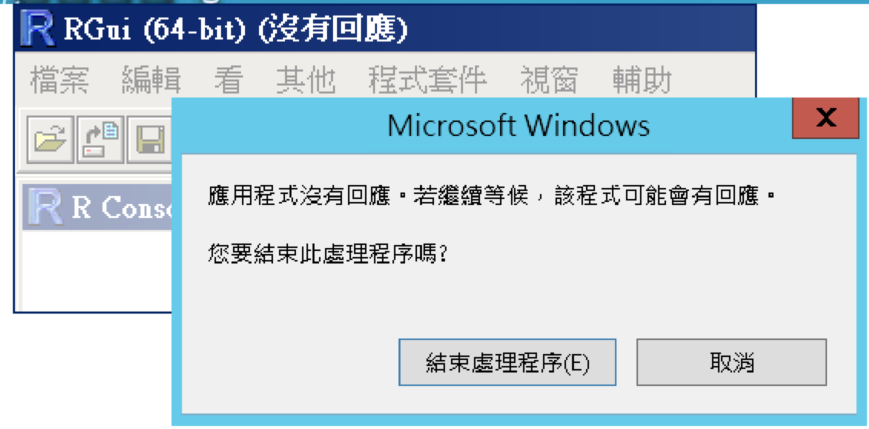
The amount of data in our world has been exploding, and analyzing large data sets—so-called big data—will become a key basis of competition, underpinning new waves of productivity growth, innovation, and consumer surplus, according to research by MGI and McKinsey's Business Technology Office. Leaders in every sector will have to grapple with

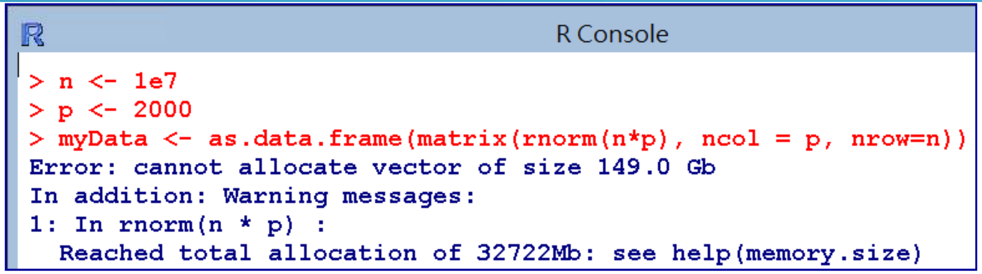
McKinsey Global Institute

McKinsey&Company

For data that is larger than the machine memory, we consider it as Big Data.

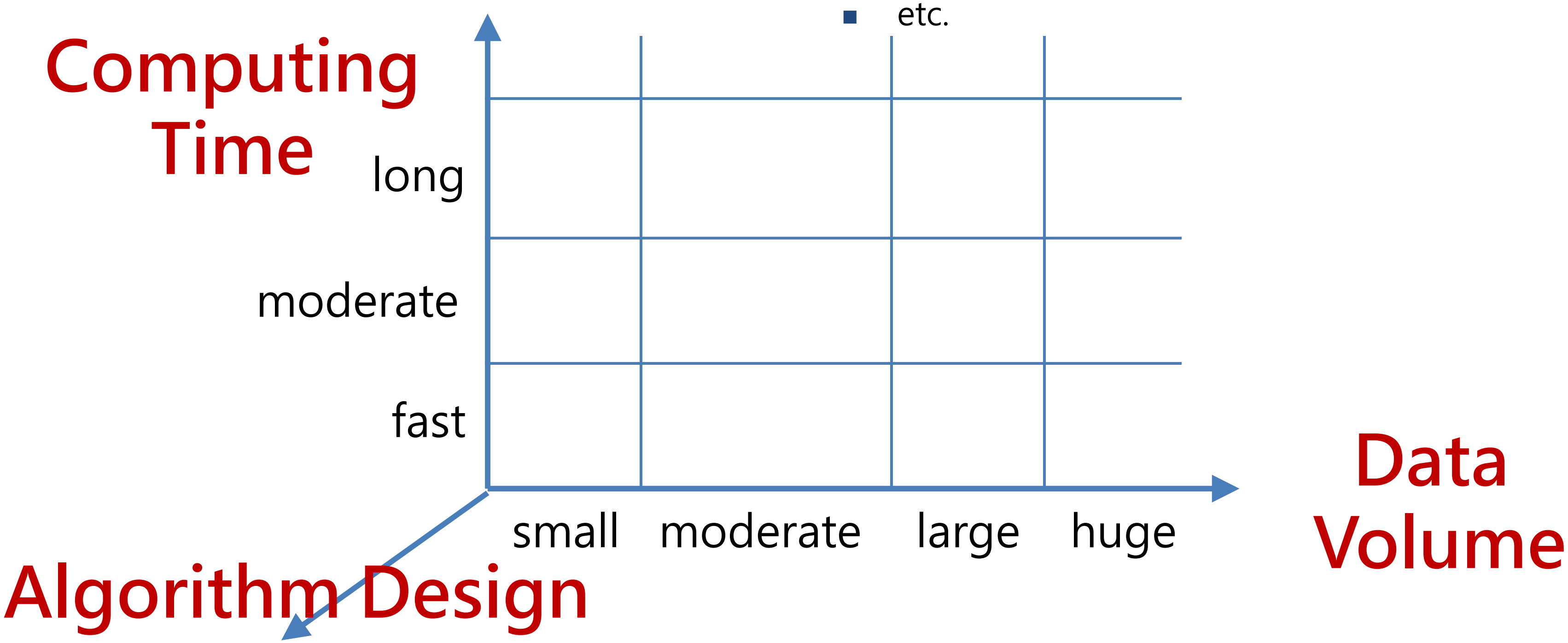
實際狀況 in R




R Console

```
> n <- 1e7
> p <- 2000
> myData <- as.data.frame(matrix(rnorm(n*p), ncol = p, nrow=n))
Error: cannot allocate vector of size 149.0 Gb
In addition: Warning messages:
1: In rnorm(n * p) :
  Reached total allocation of 32722Mb: see help(memory.size)
```

- What gets more difficult when data is big?
  - The data may not load into memory
  - analyzing the data may take a long time
  - visualizations get messy
  - etc.

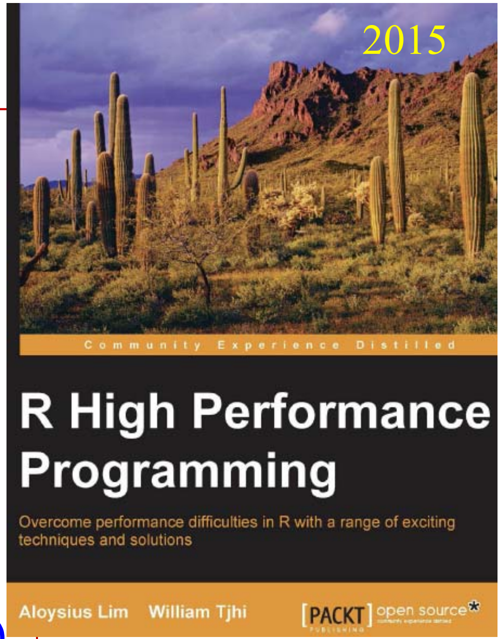


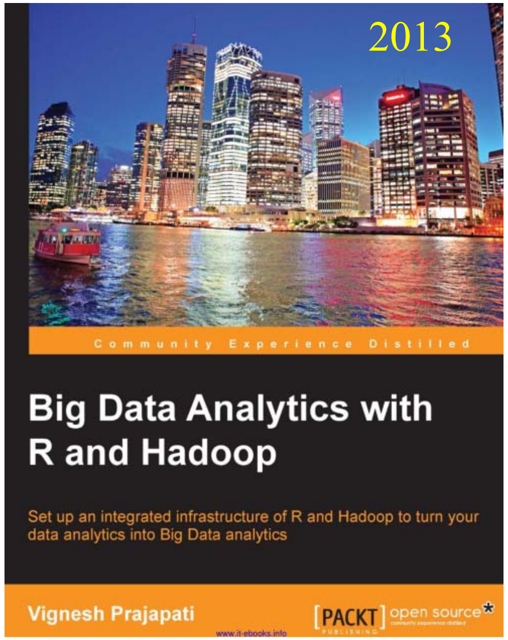
CRAN Task View: High-Performance and Parallel Computing with R

https://cran.r-project.org/web/views/HighPerformanceComputing.html

- Parallel computing: Explicit parallelism, Implicit parallelism, Grid computing, Hadoop, Random numbers, Resource managers and batch schedulers, Applications, GPUs.
- Large memory and out-of-memory data
- Easier interfaces for Compiled code
- Profiling tools

```
> version
platform       x86_64-w64-mingw32
arch           x86_64
os             mingw32
system         x86_64, mingw32
status
major          3
minor          2.2
year           2015
month          08
day            14
svn rev        69053
language       R
version.string R version 3.2.2 (2015-08-14)
nickname       Fire Safety
> Sys.getenv("R_ARCH")
[1] "/x64"
```





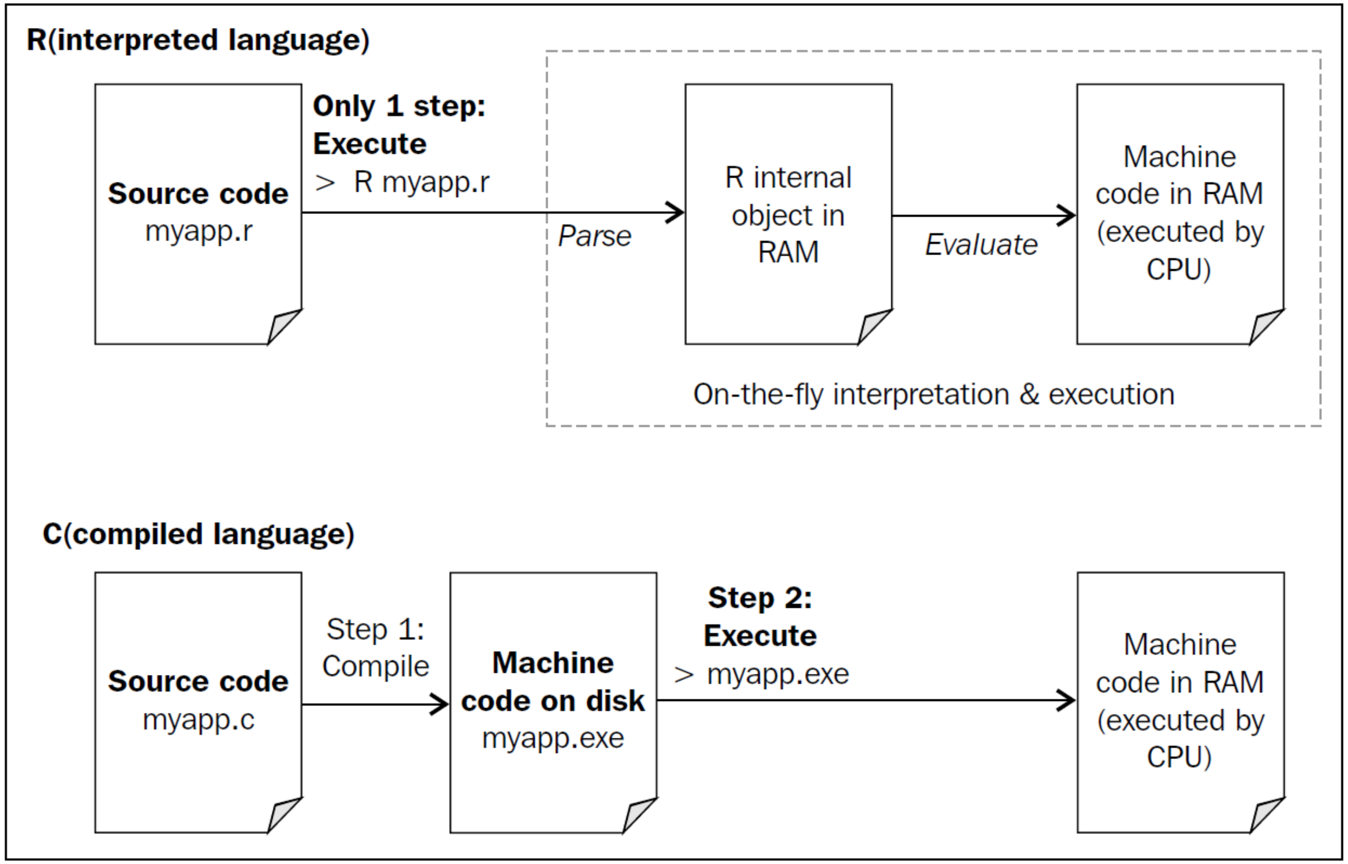
Why Are R Programs Sometimes Slow?

- Three constraints on computing performance: CPU, RAM, and disk I/O.
- R is interpreted on the fly.
- R is single-threaded.
- R requires all data to be loaded into memory.
- Algorithm design affects time and space complexity.

Source: https://www.dreamstime.com

R is Interpreted On the Fly

- R code runs relatively slow because it is reinterpreted every time you run it, even when it has not changed.
- For compiled language, once the code has been compiled, it runs very quickly on the CPU since it is already in the computer's native language.



Source: Aloysius Lim, and William Tjhi, R High Performance Programming, Packt Publishing, January 30, 2015.

Memory Allocation in R

- 當R啟動時，設定最大可獲得的記憶體:

```
"C:\Program Files\R\R-3.2.2\bin\x64\Rgui.exe" --max-mem-size=2040M
```

  - 最小需求是32MB.
  - R啟動後僅可設定更高值，不能再用**memory.limit**設定較低的值。

```
> #目前使用的記憶體量
> memory.size(max = FALSE)
[1] 3845.87
>
> #從作業系統可得到的最大量記憶體
> memory.size(max = TRUE)
[1] 3846.25
>
> #列出目前記憶體的限制
> memory.limit(size = NA)
[1] 16343
>
> #設定新的記憶體限制為 1024 MB
> memory.limit(size = 1024)
[1] 16343
Warning message:
In memory.limit(size = 1024) : 無法減少記憶體限制：已忽略
```

- R與Windows作業系統
  最大可獲得的記憶體
  - 32-bit R + 32-bit Windows: 2GB.
  - 32-bit R + 64-bit Windows: 4GB.
  - 64-bit R + 64-bit Windows: 8TB.

Report the Space Allocated for an Object: object.size{utils}

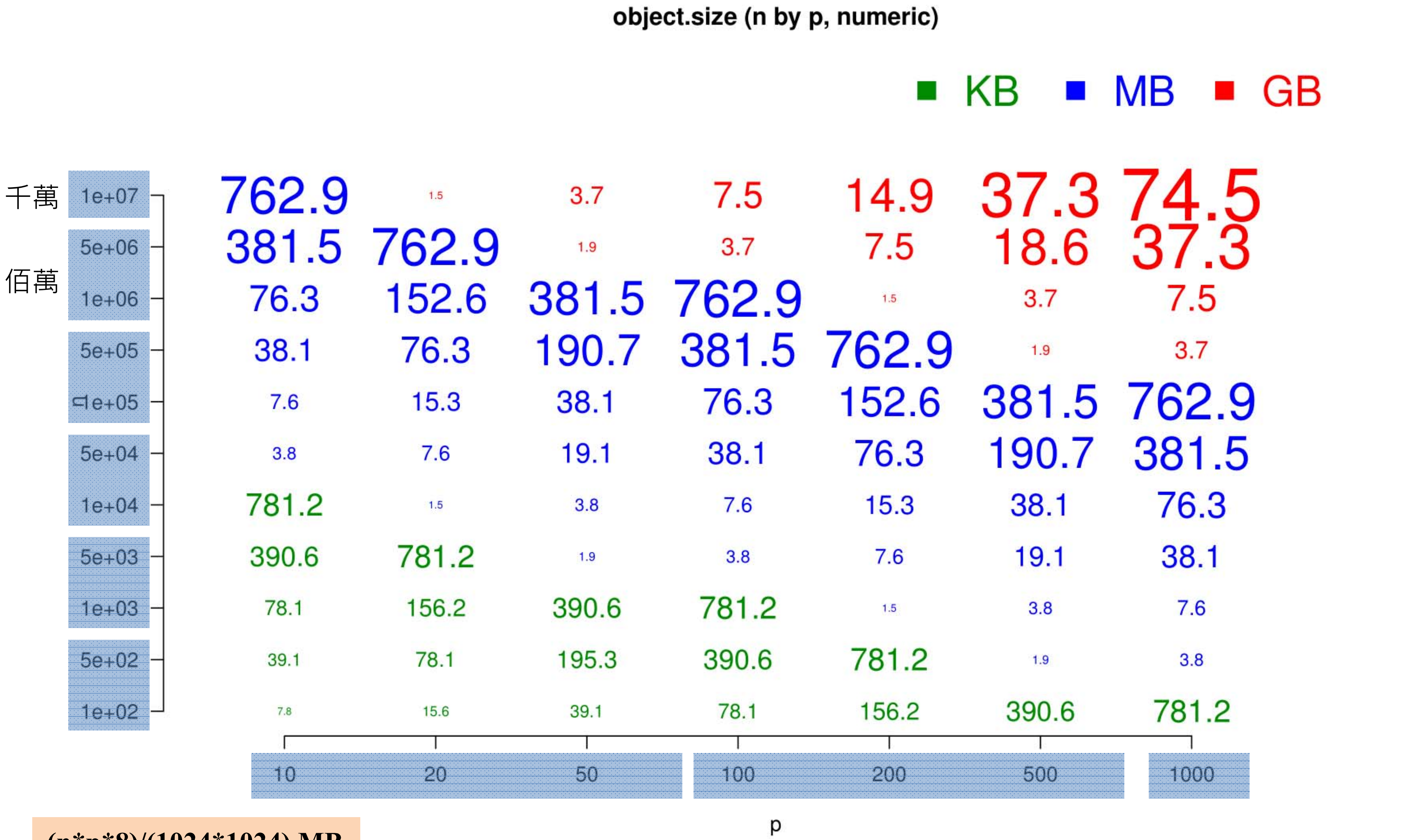
- 儲存R物件所佔用的記憶體估計。

```
object.size(x)
print(object.size(x), units = "Mb")
```

```
> n <- 10000
> p <- 200
> myData <- as.data.frame(matrix(rnorm(n*p), ncol = p, nrow=n))
> print(object.size(myData), units = "Mb")
15.3 Mb

> write.table(myData, "myData.txt") ## 約 34.7 MB

> InData <- read.table("myData.txt")
> print(object.size(InData), units = "Mb")
15.6 Mb
```

NOTE: Under any circumstances, you cannot have more than 2^{31}-1=2,147,483,647 rows or columns.

object.size{utils}



(n*p*8)/(1024*1024) MB

1 Bit = Binary Digit; 8 Bits = 1 Byte; 1024 Bytes = 1 Kilobyte; 1024 Kilobytes = 1 Megabyte
1024 Megabytes = 1 Gigabyte; 1024 Gigabytes = 1 Terabyte; 1024 Terabytes = 1 Petabyte

Measuring execution time: system.time{base}

```
myFun <- function(n){
    for(i in 1:n){
        x <- x + i
    }
    x
}
```

```
> start.time <- Sys.time()
> ans <- myFun(10000)
> end.time <- Sys.time()
> end.time -start.time
Time difference of 0.0940001 secs
```

```
> system.time({
+     ans <- myFun(10000)
+ })
   user  system elapsed
   0.04    0.00    0.05
```

See also : `microbenchmark`, `rbenchmark` packages

```
myPlus <- function(n){
    x <- 0
    for(i in 1:n){
        x <- x + sum(rnorm(i))
    }
    x
}
```

```
myProduct <- function(n){
    x <- 1
    for(i in 1:n){
        x <- x * sum(rt(i, 2))
    }
    x
}
```

```
> system.time({
+     a <- myPlus(5000)
+ })
   user  system elapsed
   3.87    0.00    3.91
> system.time({
+     b <- myProduct(5000)
+ })
   user  system elapsed
  10.36    0.00   10.42
```

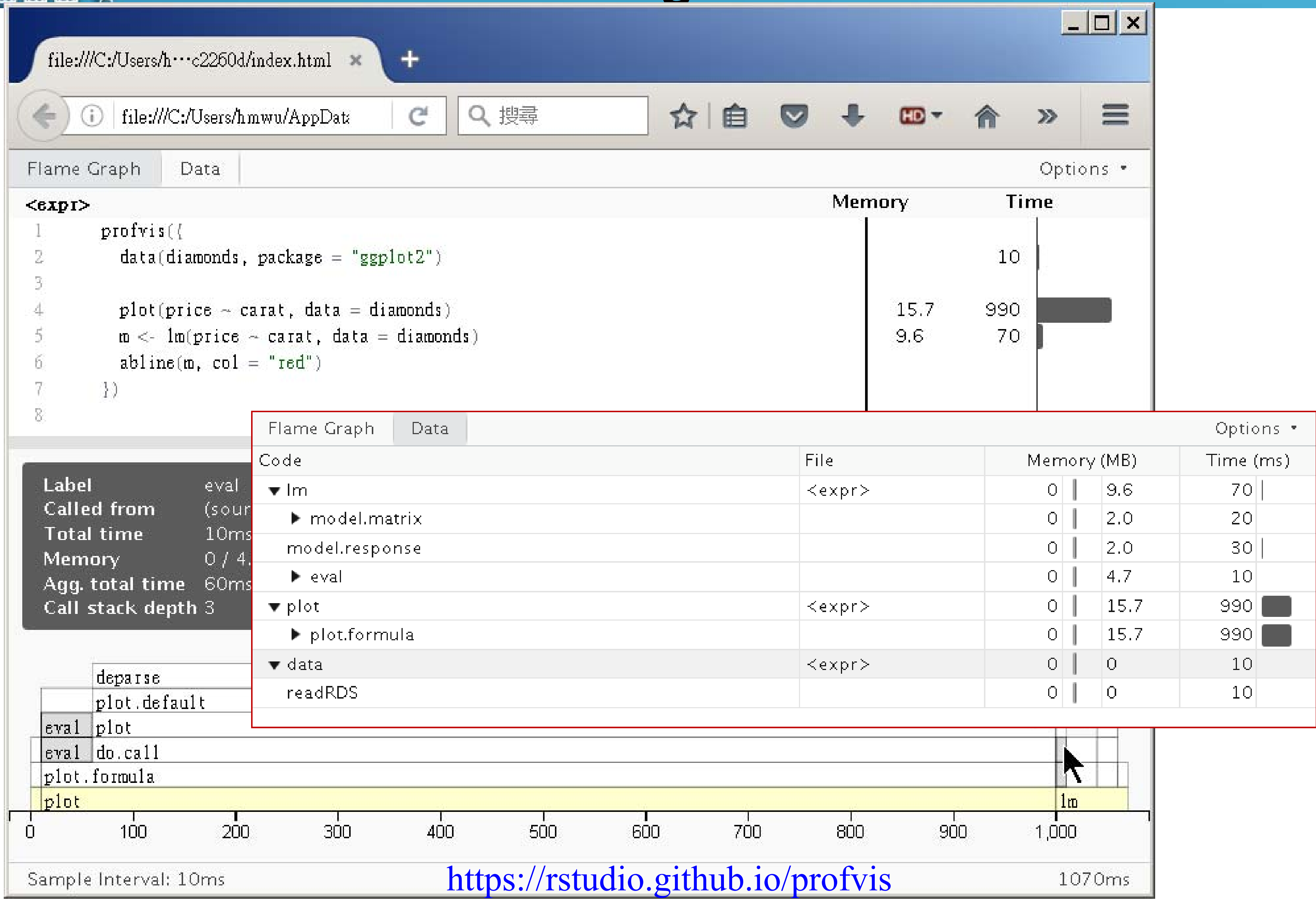
Profiling Memory Utilization: Rprof{utils}


```
myFun2 <- function(n){
    a <- myPlus(n)
    b <- myProduct(n)
    list(a, b)
}
```

```
> Rprof("Rprof-mem.out", memory.profiling = TRUE)
> ans <- myFun2(5000)
> Rprof(NULL)
> summaryRprof("Rprof-mem.out", memory = "both")
$by.self
        self.time self.pct total.time total.pct mem.total
"rt"         3.60    70.87       3.60     70.87     112.6
"rnorm"      1.44    28.35       1.44     28.35     108.8
"sum"        0.04     0.79       0.04      0.79       2.0

$by.total
            total.time total.pct mem.total self.time self.pct
"myFun2"          5.08    100.00     223.5      0.00     0.00
"myProduct"       3.62     71.26     113.2      0.00     0.00
"rt"              3.60     70.87     112.6      3.60    70.87
"myPlus"          1.46     28.74     110.3      0.00     0.00
"rnorm"           1.44     28.35     108.8      1.44    28.35
"sum"             0.04      0.79       2.0      0.04     0.79

$sample.interval
[1] 0.02

$sampling.time
[1] 5.08
```

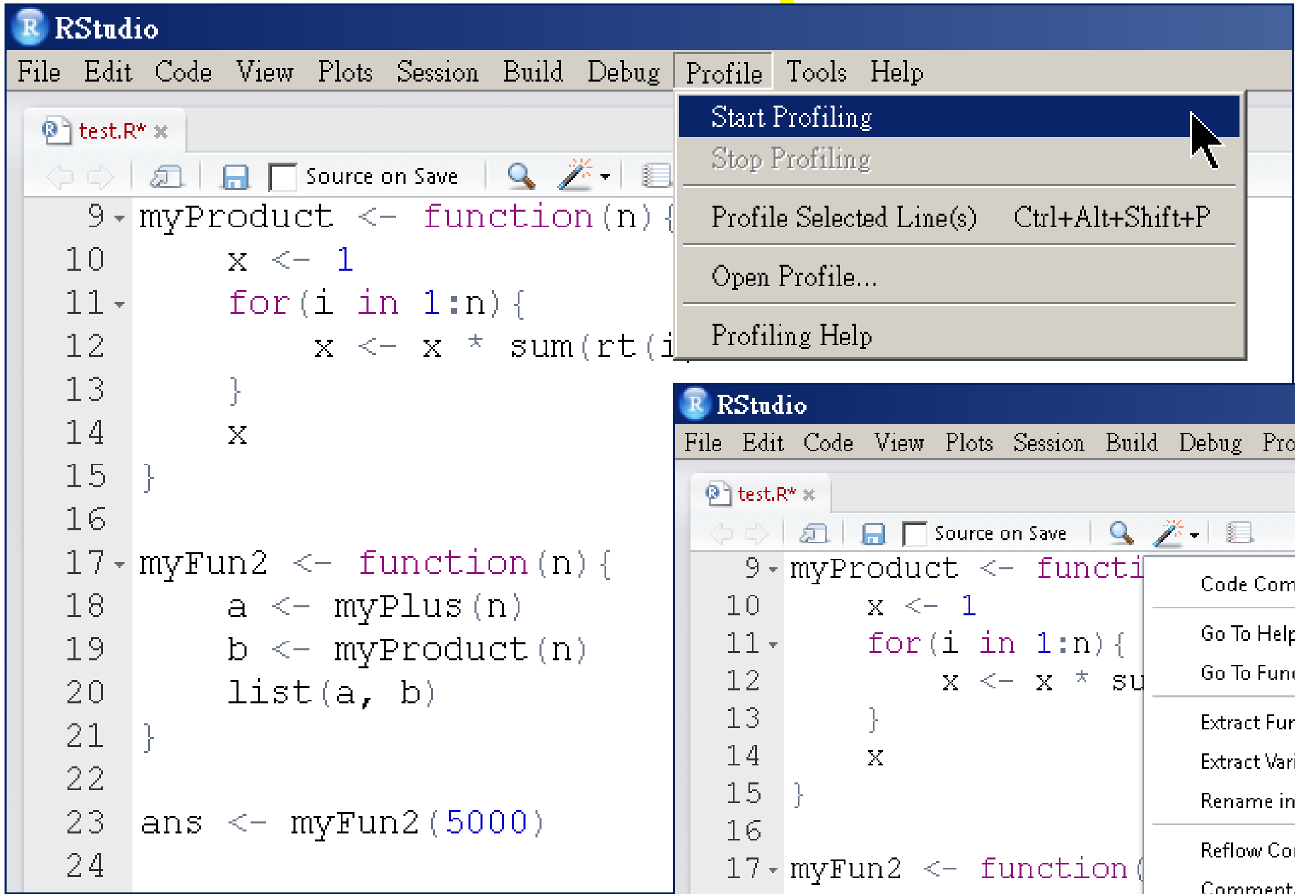
<See also the **proftools** package>

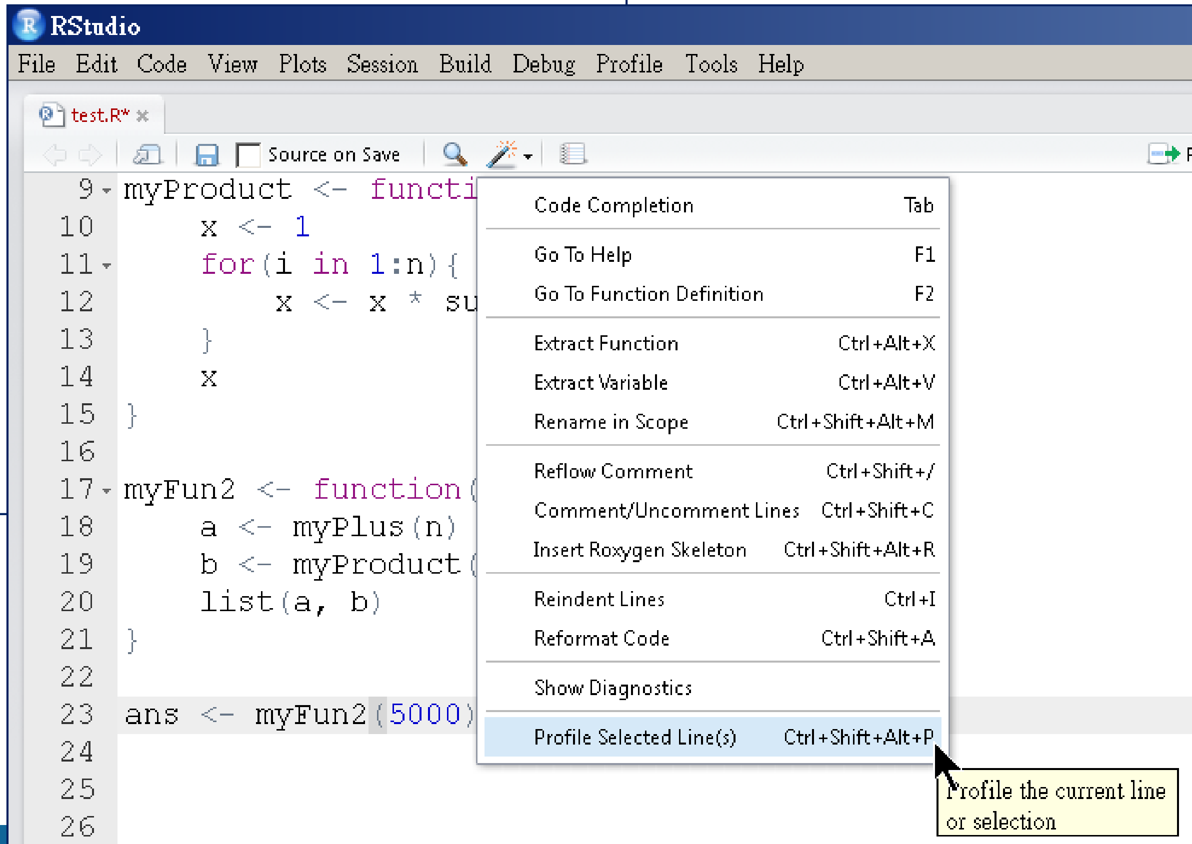
profvis: Interactive Visualizations for Profiling R Code



https://rstudio.github.io/profvis

Profiling with RStudio and profvis

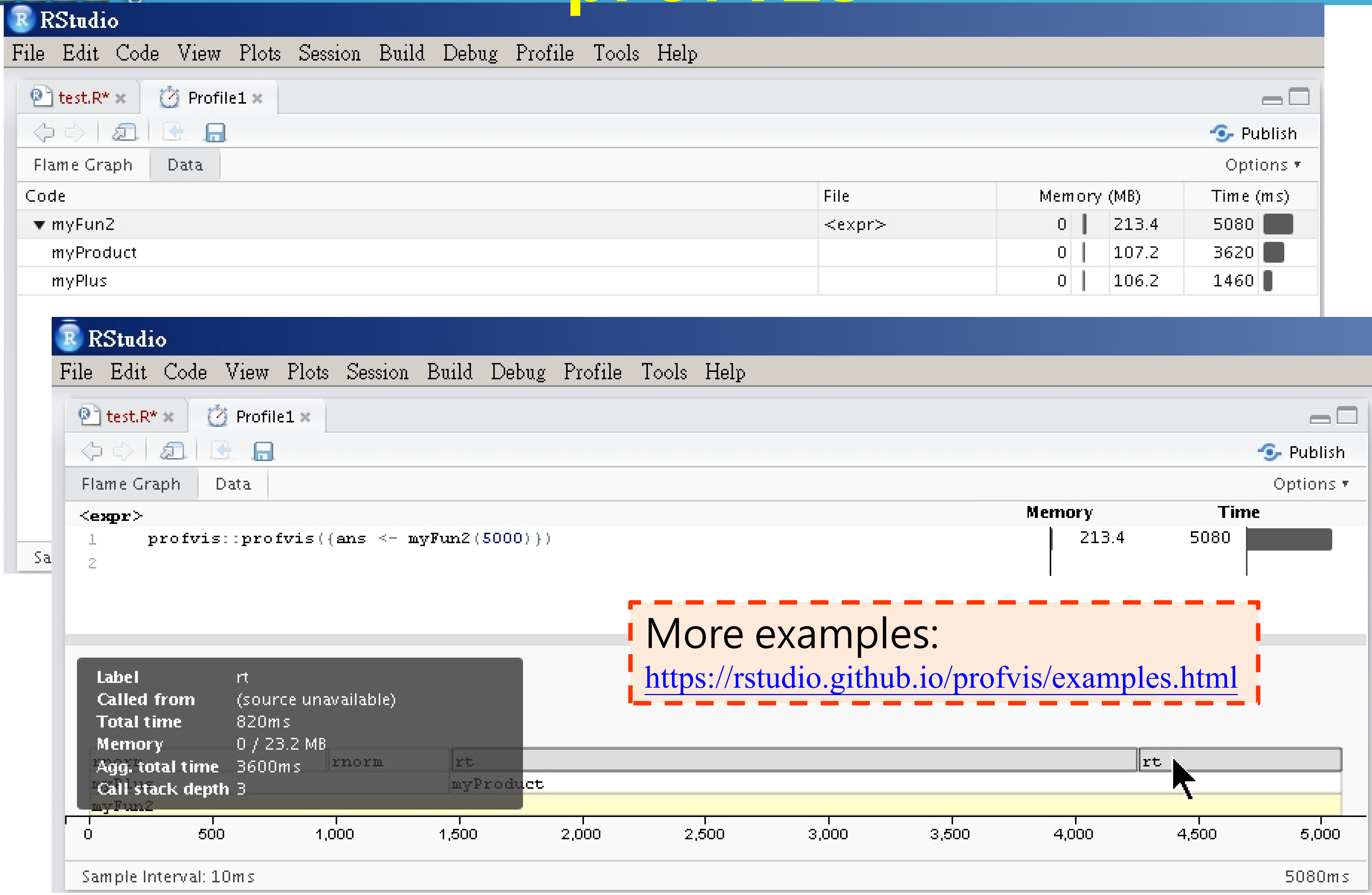




RStudio v0.99.1208 Preview (2016-06-03)

https://rstudio.github.io/profvis/

Profiling with RStudio and profvis



More examples:
https://rstudio.github.io/profvis/examples.html

(I) Speed Up R

(A) Simple Tweaks to Make R Run Faster

- (A1) Vectorization
- (A2) Use of built-in functions
- (A3) Preallocating memory
- (A4) Use of simpler data structures
- (A5) Use of hash tables for frequent lookups on large data
- (A6) Seeking fast alternative packages in CRAN

- These tweaks should be taken as the first steps in order to optimize an R code. (養成良好程式設計習慣及風格)

(A1) Vectorization

- In essence, vectorization allows R operators to take vectors as arguments for quick processing of multiple values.

```
> # Non-vectorized
> n <- 1E6
> mydata <- rnorm(n)
> system.time({
+     s1 <- 0
+     for(j in 1:n){
+         s1 <- s1 + mydata[j]^2
+     }
+     s1
+ })
   user  system elapsed
   0.97    0.00    0.97
```

```
> #Vectorized
> system.time(s2 <- sum(mydata^2))
   user  system elapsed
      0       0       0
```

Vector size	100,000	1,000,000	10,000,000	100,000,000
Non-vectorized	120 ms	1.19 s	11.9 s	117 s
Vectorized	508 µs	5.67 ms	52.5 ms	583 ms

Source: Aloysius Lim, and William Tjhi, R High Performance Programming, Packt Publishing, January 30, 2015.

(A2) Use of Built-in Functions

- R and some CRAN packages provide a rich set of functions that are implemented in compiled languages such as C/C++.

```
> # use apply
> n <- 1E4
> p <- 1000
> mydata <- matrix(rnorm(n*p), nrow=n, ncol=p)
> system.time(data_sum1 <- apply(mydata, 1, sum))
   user  system elapsed
   0.30    0.02    0.31
> # rowSums is an optimized and precompiled C function
> system.time(data_sum2 <- rowSums(mydata))
   user  system elapsed
   0.04    0.00    0.03
```

- Use the optimized BLAS (Basic Linear Algebra Subprograms) libraries (the Mac OS X version of R comes enabled with the optimized BLAS.)

```
data <- rnorm(1E7)
dim(data) <- c(1E4, 1E3)
system.time(data_mul <- t(data) %*% data)
##  user  system elapsed
## 7.123   0.015   7.136
system.time(data_mul <- t(data) %*% data) # with optimized BLAS
##  user  system elapsed
## 1.304   0.005   0.726
```

Source: Aloysius Lim, and William Tjhi, R High Performance Programming, Packt Publishing, January 30, 2015.

(A3) Preallocating Memory

- Dynamic memory allocation slows down a program. Every time a vector is resized, the program needs to perform extra steps that include copying the vector to a larger or smaller memory block and deleting the old vector. These steps are not needed if the memory is preallocated.

```
> n <- 1E4; dataV1 <- 1
> system.time({
+     for (j in 2:n) {
+         dataV1 <- c(dataV1, dataV1[j-1] + sample(-5:5, size=1))
+     }
+ })
   user  system elapsed
   0.20    0.00    0.21
>
> dataV2 <- numeric(n)
> dataV2[1] <- 1
> system.time({
+     for (j in 2:n) {
+         dataV2[j] <- dataV2[j-1] + sample(-5:5, size=1)
+     }
+ })
   user  system elapsed
   0.07    0.00    0.06
```

Vector size	10	100	1000	10,000
Dynamic allocation	0	0.006	0.288	25.373
Preallocated	0.001	0.006	0.062	0.577

Source: Lim and Tjhi, R High Performance Programming, Packt Publishing, January 30, 2015.

(A4) Use of Simpler Data Structures

- A data.frame object allows more flexibility than a matrix by allowing variables of different types.
- Applying a matrix operation on a data.frame is slower than on a matrix. One of the reasons is that most matrix operations first coerce the data.frame into a matrix before performing the computation.

```
> # working on matrix
> n <- 1E4
> p <- 1000
> mydata.mt <- matrix(rnorm(n*p), nrow=n, ncol=p)
> system.time(rs1 <- rowSums(mydata.mt))
   user  system elapsed 
   0.03    0.00    0.03 
> # working on data.frame
> mydata.df <- data.frame(mydata.mt)
> system.time(rs2 <- rowSums(mydata.df))
   user  system elapsed 
   0.08    0.00    0.08 
```

```
> system.time(mydata.df[mydata.df$X100 > 0 & mydata.df$X200 < 0,])
   user  system elapsed 
   0.72    0.05    0.76 
> system.time(mydata.df[which(mydata.df$X100 > 0 & mydata.df$X200 < 0),])
   user  system elapsed 
   0.25    0.00    0.25 
```

(A5) Use of Hash Tables for Frequent Lookups on Large Data

- The implementation of `lists` in R is not optimized for lookup; it incurs O(N) time complexity to perform a lookup on a list of N elements.
- A hash table's (雜湊表) lookup incurs O(1) time complexity. (R package `hash`)

```
> n <- 1E6
> mydata <- rnorm(n)
> mydata.ls <- as.list(mydata)
> names(mydata.ls) <- paste("X", c(1:n), sep="")
> head(mydata.ls, 3)
$X1
[1] -0.1038026

$X2
[1] -0.09553649

$X3
[1] -0.7474468

> library(hash)
> mydata.hs <- hash(names(mydata.ls), mydata)
> str(mydata.hs)
Formal class 'hash' [package "hash"] with 1 slot
  ..@ .xData:<environment: 0x00000000560a3fe8>
>
> #create lookups
> id <- sample(1:n, size=1000, replace=T)
> lookups <- paste("X", id, sep="")
```

雜湊表(hash table)是一種可以快速處理資料新增、搜尋及刪除的資料結構。利用資料的鍵值(key)直接對應至儲存位置的方法，雜湊表可以在幾次的資料比對後就完成資料加入、搜尋及刪除的動作。

```
> #comparison
> comptime.list <- sapply(lookups,
+     FUN = function(x){
+         system.time(mydata.ls[[x]])[3]}
+ )
> sum(comptime.list)
[1] 8.97
>
>
> comptime.hash <- sapply(lookups,
+     FUN = function(x){
+         system.time(mydata.hs[[x]])[3]}
+ )
> sum(comptime.hash)
[1] 0.17
```

(A6) Seeking Fast Alternative Packages in CRAN

- `fastcluster`: using an optimized C++ code that improves the speed significantly compared to the routines implemented in `hclust`.
- `fastmatch`: a faster version of base R's `match` function
- `RcppEigen`: a faster version of linear modeling `lm`
- `data.table`: This offers faster data manipulation operations compared to the standard `data.frame` operations
- `dplyr`: This offers a set of tools to manipulate data frame-like objects efficiently

```
> n <- 1E4
> p <- 100
> mydata <- matrix(rnorm(n*p), nrow=n, ncol=p)
> mydata.dist <- dist(mydata)
> system.time(hc.obj1 <- hclust(mydata.dist))
   user  system elapsed
   4.46    0.20    4.66
>
> library(fastcluster)
> system.time(hc.obj2 <- hclust(mydata.dist))
   user  system elapsed
   2.59    0.11    2.70
```

See also: http://www.rdocumentation.org/

(B) Using Compiled Code for Greater Speed

(B1) Compiling R code before execution.

- **`compiler`** package: allow to compile R code beforehand and save R a step or two when we execute the code.
- **`jit`** package: enable just-in-time (JIT) compilation of a block of R code.

(B2) Using compiled languages such as C and use them from within R.

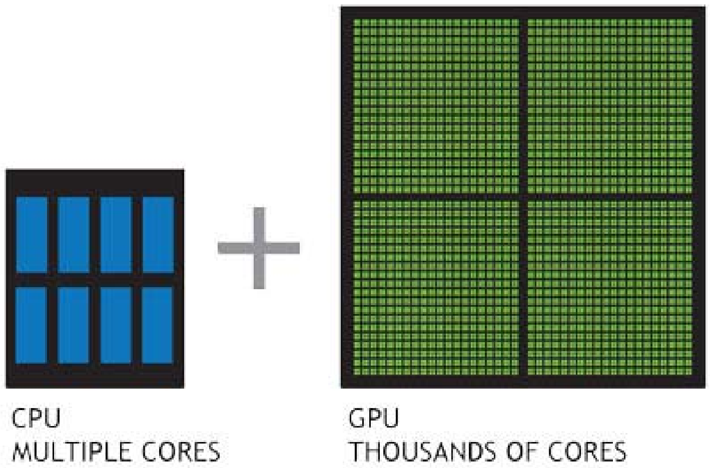
- **`inline`** package : allows to embed C, C++, Objective-C, Objective-C++, and Fortran code within R.

(B3) Calling external compiled code

- R provides a few interfaces to call the external compiled code: **`.C()`**, **`.Fortran()`**, **`.Call()`**, **`.External()`**
- **`Rcpp`** package : provides a convenient, higher-level API to the **`.Call()`** interface for C++ code.

(C) Using GPUs to Run R Even Faster

- Graphics Processing Unit (GPU), known as a graphics card.
- To achieve real-time rendering, most GPU computations are done in a highly parallel manner, with many more cores than CPUs - a modern GPU might have more than 2,000 cores. Given that one core can run multiple threads, it is possible to run tens of thousands of parallel threads on a GPU.
- Will need an NVIDIA GPU with CUDA capabilities.
  - CUDA(Compute Unified Device Architecture)是NVIDIA 的平行運算架構，可運用繪圖處理單元(GPU) 的強大處理能力，大幅增加運算效能。



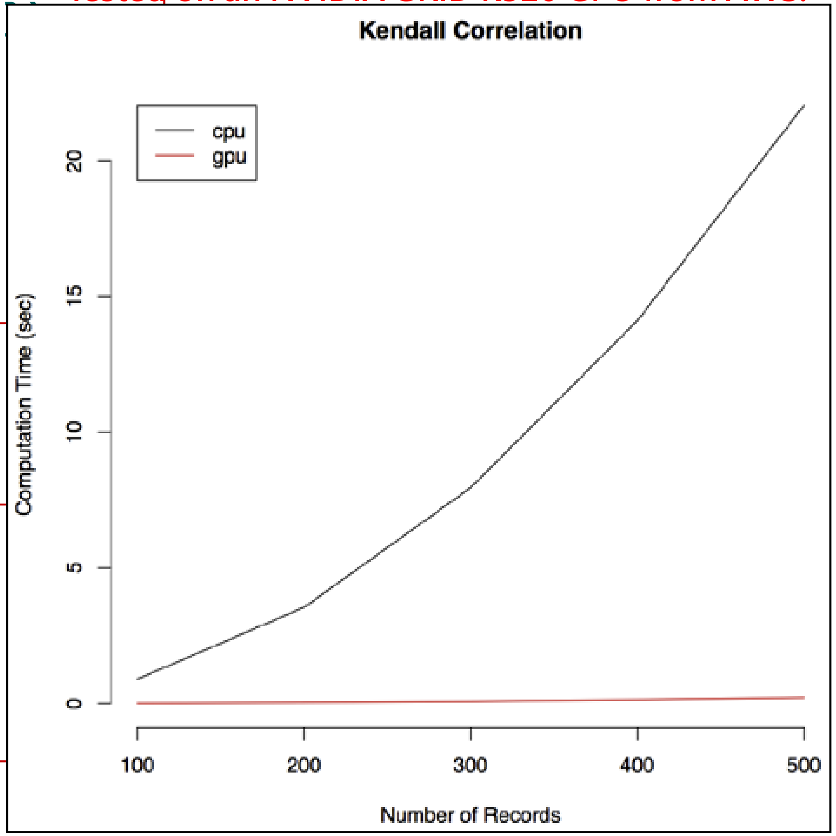
http://www.nvidia.com.tw/object/what-is-gpu-computing-tw.html

Example of Using GPUs to Run R

- Calculate Kendall correlations on random datasets having 100 variables with a varying number of observations from 100, 200, … to 500 records in order to observe the speedup in comparison to the CPU version.

```
library(gputools)
A <- lapply(c(1:5), function(x){
    matrix(rnorm((x*1E2) * 1E2), 1E2, (x*1E2))

cpu.k.time <- sapply(A, function(x){
    system.time(cor(x, "kendall"))[[3]]})

gpu.k.time <- sapply(A, function(x) {
    system.time(gpuCor(x, "kendall"))[[3]]})
```

Tested on an NVIDIA GRID K520 GPU from AWS.




```
> str(A)
List of 5
 $ : num [1:100, 1:100] 0.0318 1.3748 -0.0756 -0.1787 -0.4244 .
 $ : num [1:100, 1:200] -0.7577 -0.0139 0.8256 1.1097 0.1976 ..
 $ : num [1:100, 1:300] -0.838 0.398 0.934 0.13 1.071 ...
 $ : num [1:100, 1:400] 0.71 1.9 0.975 -0.341 1.686 ...
 $ : num [1:100, 1:500] -0.89 -1.378 -0.517 -0.552 1.69 ...
```

R Packages Related to GPUs

- **gputools**: provides some data-mining algorithms which are implemented using a mixture of nVidia's CUDA langauge and cublas library.
- **cudaBayesreg**: implements the rhier Linear Model from the bayesm package using nVidia's CUDA langauge and tools to provide high-performance statistical analysis of fMRI voxels.
- **rgpu**: aims to speed up bioinformatics analysis by using the GPU.
- **gcbd**: implements a benchmarking framework for BLAS and GPUs.
- **OpenCL**: provides an interface from R to OpenCL permitting hardware- and vendor neutral interfaces to GPU programming.
- **HiPLARM**: provide High-Performance Linear Algebra for R using multi-core and/or GPU support using the PLASMA / MAGMA libraries from UTK, CUDA, and accelerated BLAS.
- **gmatrix**: enables the evaluation of matrix and vector operations using GPU coprocessors such that intermediate computations may be kept on the coprocessor and reused, with potentially significant performance enhancements by minimizing data movement.
- **gpuR**: offers GPU-enabled functions for mirroring typical R syntax without the need to know OpenCL.

Factors that Affect the GPU's Performance

- GPUs work best for data parallel problems.
  - They are not suited for tasks that require large amounts of synchronization between threads.
- GPU's performance depends on the amount of data transferred between RAM and the GPU's memory.
  - The connection between the RAM and GPU's memory has a low bandwidth.
  - Good GPU programming should minimize this data transfer.
- 缺點:
  - Addressing these factors requires programming in the low-level GPU interfaces provided by RCUDA or OpenCL.
  - Packages 安裝稍為複雜。需硬體配合。
  - 目前較少分析分法: **`gputools`**: **`gpuLm, gpuGlm, gpuCor, gpuDist, gpuHclust, ...`**
  - 不一定比CPU快。

(II) Use Less RAM

(A) Simple Tweaks to Use Less RAM

(A1) Reusing objects without taking up more memory

- the copy-on-modification semantics of R's memory management. (`b <- a` # appears to make a copy of a and refer to it as b)

(A2) Removing intermediate data when it is no longer needed

(A3) Calculating values on the fly instead of storing them persistently

- On the fly computations produce intermediate data without creating variables that persist in the memory.
- Functions are a useful way to group related operations and automatically remove temporary variables when exiting the functions.

(A4) Swapping active and nonactive data

- save data to the disk to free up memory and reload them later when needed.

(B) Processing Large Datasets with Limited RAM

(B1) Using memory-efficient data structures and smaller data types

(B2) Sparse matrices

(B3) Symmetric matrices

(B4) Bit vectors

(B5) Using memory-mapped files and processing data in chunks

- R packages:
  - **bigmemory**: The Bigmemory Project (Working with Massive Matrix-like Objects in R): **biganalytics**, **bigtabulate**, **synchronicity**, **bigalgebra**. (http://www.bigmemory.org)
  - **biglm**: bounded memory linear and generalized linear models for data too large to fit in memory.
  - **ff**:
    - supports more data types than **bigmemory**.
    - provides a more data frame-like memory-mapped format is required while dealing with heterogeneous data types.
    - memory-efficient storage of large data on disk and fast access functions.

(B1) Using Memory-efficient Data Structures and Smaller Data Types

Memory-efficient Data Structures

```
> print(object.size(logical(1e6)), units="MB")
3.8 Mb
> print(object.size(integer(1e6)), units="MB")
3.8 Mb
> print(object.size(numeric(1e6)), units="MB")
7.6 Mb
> print(object.size(complex(1e6)), units="MB")
15.3 Mb
> print(object.size(rep.int(NA_character_, 1e6)), units="MB")
7.6 Mb
> print(object.size(raw(1e6)), units="MB")
1 Mb
> print(object.size(vector("list", 1e6)), units="MB")
7.6 Mb
```

Smaller data types

```
> object.size(as.numeric(seq_len(1e6)))
8000040 bytes
> object.size(as.integer(seq_len(1e6)))
4000040 bytes
> strings <- rep.int(formatC(seq_len(1e4), width = 1000), 100)
> object.size(strings)
18480040 bytes
> factors <- factor(strings)
> object.size(factors)
14560400 bytes
```

(B2) Sparse Matrices

```
> library(Matrix)
> n <- rnorm(1e6)
> n[sample.int(1e6, 7e5)] <- 0
> m.dense <- Matrix(n, 1e3, 1e3, sparse = FALSE)
> m.sparse <- Matrix(n, 1e3, 1e3, sparse = TRUE)
> object.size(n)
8000040 bytes
> object.size(m.dense)
8001112 bytes
> object.size(m.sparse)
3605424 bytes
```

Sparse data: it contains a lot of zeroes or empty values.

```
> m.dense[1:3,1:3]
3 x 3 Matrix of class "dgeMatrix"
           [,1] [,2]      [,3]
[1,]  0.0000000    0 0.0000000
[2,] -0.7627943    0 0.0000000
[3,]  0.0000000    0 0.2869535
> m.sparse[1:3,1:3]
3 x 3 sparse Matrix of class "dgCMatrix"

[1,]  .          . .
[2,] -0.7627943 . .
[3,]  .          . 0.2869535
```

Sparse matrices are also very useful for binary data (TRUE/FALSE, 0/1, "yes"/"no", "hot"/"cold", and so on).

```
> logicalData <- sample(c(FALSE, TRUE), 1e6, TRUE, c(0.7, 0.3))
> m2.dense <- Matrix(logicalData, 1e3, 1e3, sparse = FALSE)
> m2.sparse <- Matrix(logicalData, 1e3, 1e3, sparse = TRUE)
> object.size(logicalData)
4000040 bytes
> object.size(m2.dense)
4001112 bytes
> object.size(m2.sparse)
2410576 bytes
```

The sparse logical matrix is even more compact than the sparse numeric matrix,

(B3) Symmetric Matrices

- The `Matrix` package provides the `dspMatrix` class to efficiently store symmetric matrices and other efficient matrix-type data structures including triangular matrices and diagonal matrices.
  - The package makes it such that basic matrix operations, such as matrix multiplication (`%*%`), are applicable for both dense and sparse matrices.

```
> library(Matrix)
> data <- matrix(rnorm(1E5), 1E2, 1E3)
> A <- cor(data)
> isSymmetric(A)
[1] TRUE
> B <- as(A, "dspMatrix")
> object.size(A)
8000200 bytes
> object.size(B)
4005320 bytes
```

(B4) Bit Vectors

- Unlike logical values (Binary data) in R that take up four bytes (32 bits), bit vectors store each logical value using only one bit.
  - This reduces the memory consumption of logical values by a factor of 32.
  - Bit vectors cannot store the NA value, so they are not suitable for data that contains the NA values.
- **bit** package:
  - When dealing with large amounts of logical or binary data, bit vectors not only save memory but also provide a speed boost when they are operated on.

```
> # compare the sizes of a logical vector and the equivalent bit vector
> library(bit)
> a <- sample(c(TRUE, FALSE), 1e6, TRUE)
> object.size(a)
4000040 bytes
> b <- as.bit(a)
> object.size(b)
126344 bytes
```

(B5) Using Memory-mapped Files and Processing Data in Chunks

- Some datasets are still too large to fit in or be processed in the memory. One way to work with such large data is to store them on a disk in the form of memory-mapped files and load the data into the memory for processing one small chunk at a time.
- Some algorithms can easily be converted to compute on chunks of data, while others might require substantial effort to do so.

```
> setwd("D:/test-R")
> library(bigmemory)
> #object bm.obj stores a pointer to the new memory-mapped file bm.
> bm.obj <- big.matrix(1e9, 3, backingfile = "bm", backingpath = getwd())
> bm.obj
An object of class "big.matrix"
Slot "address":
<pointer: 0x000000000fd201b0>
```

the new file bm has a size of 22 GB
#"bm.desc" was created to describe the data file.
This is used to retrieve the memory-mapped file at a later time such as

```
my.bm <- attach.big.matrix("bm.desc")
```

Example: A Chunked Computation

```
#fills in bm.obj with random numbers in 100 chunks of 10 million rows at a
time
chunksize <- 1e7
start <- 1
while (start <= nrow(bm.obj)) {
    end <- min(start + chunksize - 1, nrow(bm.obj))
    chunksize <- end - start + 1
    bm.obj[start:end, 1] <- rpois(chunksize, 1e3)
    bm.obj[start:end, 2] <- sample(0:1, chunksize, TRUE, c(0.7, 0.3))
    bm.obj[start:end, 3] <- runif(chunksize, 0, 1e5)
    start <- start + chunksize
}
```

```
> dim(bm.obj)
[1] 1e+09 3e+00

> head(bm.obj)
     [,1] [,2]      [,3]
[1,] 1007    1 31038.563
[2,]  970    0  4738.024
[3,] 1027    0 65442.558
[4,] 1012    0 67877.496
[5,] 1027    0 71838.828
[6,] 1032    0 29757.398
```

When the subsetting operator `[,]` is used, **`bigmemory`** automatically loads the relevant portions of the data into the RAM and removes portions that are no longer needed.

A Chunked Computation to Find the Standard Deviation of Each Column


```
> col.sums <- numeric(3)
> chunksize <- 1e7
> start <- 1
> while (start <= nrow(bm.obj)) {
+     end <- min(start + chunksize - 1, nrow(bm.obj))
+     col.sums <- col.sums + colSums(bm.obj[start:end, ])
+     start <- start + chunksize
+ }
> col.means <- col.sums/nrow(bm.obj)
> col.means
[1]  1000.001289     0.299985 49999.209345
```

Calling `sd(bm[1, ])` might not work, as even a single column of data can exceed available memory.

```
> col.sq.dev <- numeric(3)
> start <- 1
> while (start <= nrow(bm.obj)) {
+     end <- min(start + chunksize - 1, nrow(bm.obj))
+     sq.dev <- sq.dev + rowSums((t(bm.obj[start:end, ])- col.means)^2)
+     start <- start + chunksize
+ }
> col.var <- sq.dev/(nrow(bm.obj) - 1)
> col.sd <- sqrt(col.var)
> col.sd
[1] 31.623471   0.458251   28867.409206
```

```
> library(biganalytics)
> colsd(bm.obj)
[1]  31.623471  0.458251  28867.409206
```

Example: The Bigmemory Project

- 例子: Airline on-time performance data from the 2009 JSM Data Expo
- 資料大小: airline.csv: 約 11 GB (120 million rows and 29 columns).
- 運算平台: A laptop with only 4 GB of RAM
- 資料讀取結果:

```
> library(bigmemory)
> library(biganalytics)
> x <- read.big.matrix("airline.csv", type="integer", header=TRUE,
   ...) #約 25分鐘。
> summary(x) #約 3-4 分鐘
```

- 統計分析結果:

```
# lm() 需要超過10GB的記憶體
> blm <- biglm.big.matrix(ArrDelay ~ Age + Year, data=x)
# 3分鐘 + 幾百MB記憶體
> summary(blm)
Large data regression model: biglm(formula = formula, data = data,
   ...)
Sample size = 84216580
...
```

- 結論: 不是每個統計方法in R都有相對應的big版本。

http://www.bigmemory.org

(III) Multiplying Performance with Parallel Computing

平行化計算?

Amdahl's Law (阿姆達爾定律)

- Most algorithms fall somewhere in between with some steps that must run serially and some that can run in parallel.
- Amdahl's law provides a way to estimate the best attainable performance gain when you convert a code from serial to parallel execution:

$$T(n) = T(1)(P + (1-P)/n)$$

  - $T(n)$ is the time taken to execute the task using n parallel processes
  - P is the proportion of the whole task that is strictly serial
- The theoretical best possible speed up of the parallel algorithm is:

$$S(n) = T(1) / T(n) = 1 / (P + (1-P)/n)$$

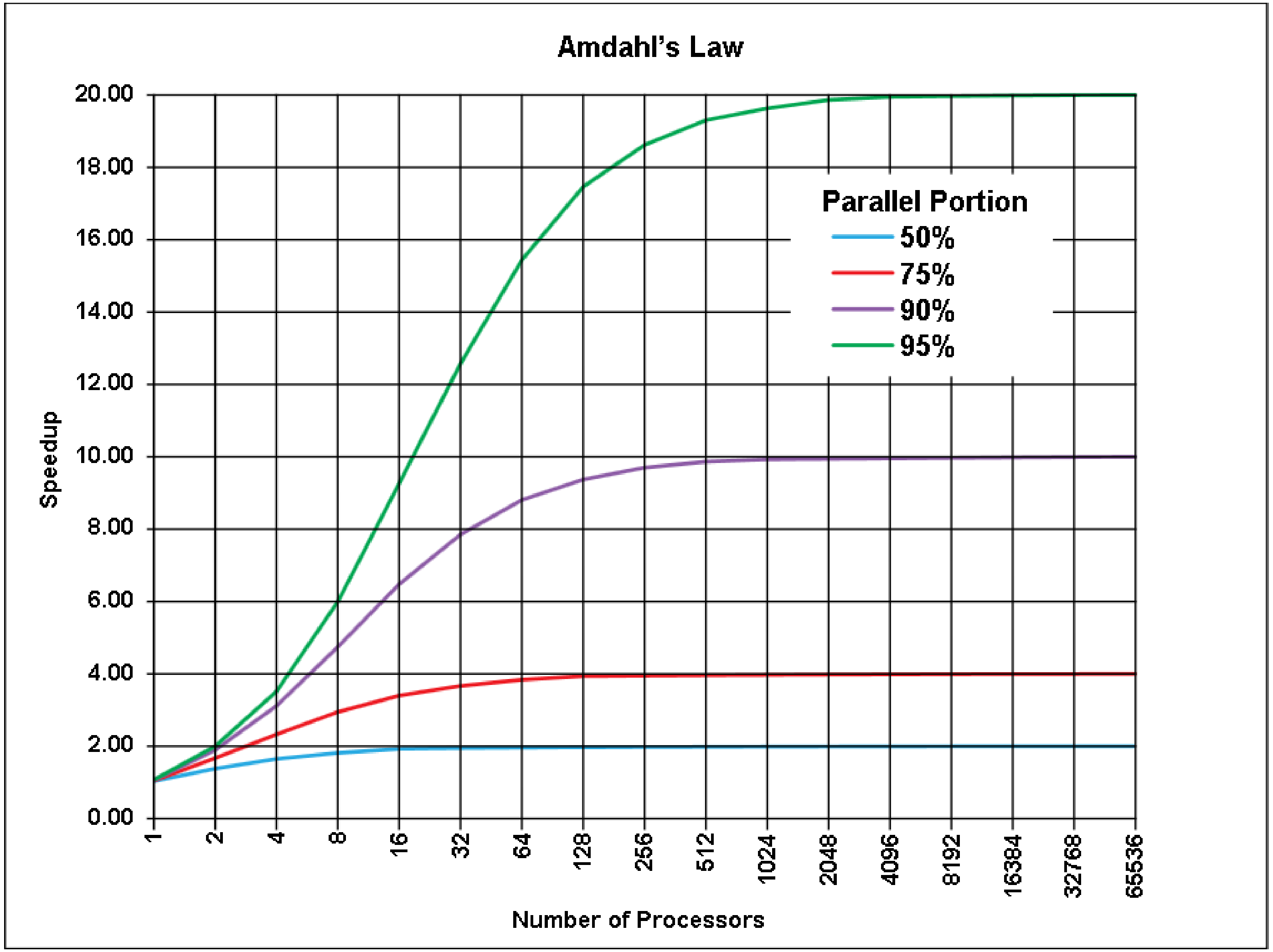
Example: given a task that takes 10 seconds to execute on one processor, where half of the task can be run in parallel, then the best possible time to run it on four processors is

$$T(4) = 10(0.5 + (1-0.5)/4) = 6.25 \text{ seconds.}$$

The theoretical best possible speed up of the parallel algorithm with four processors is

$$1 / (0.5 + (1-0.5)/4) = 1.6x.$$

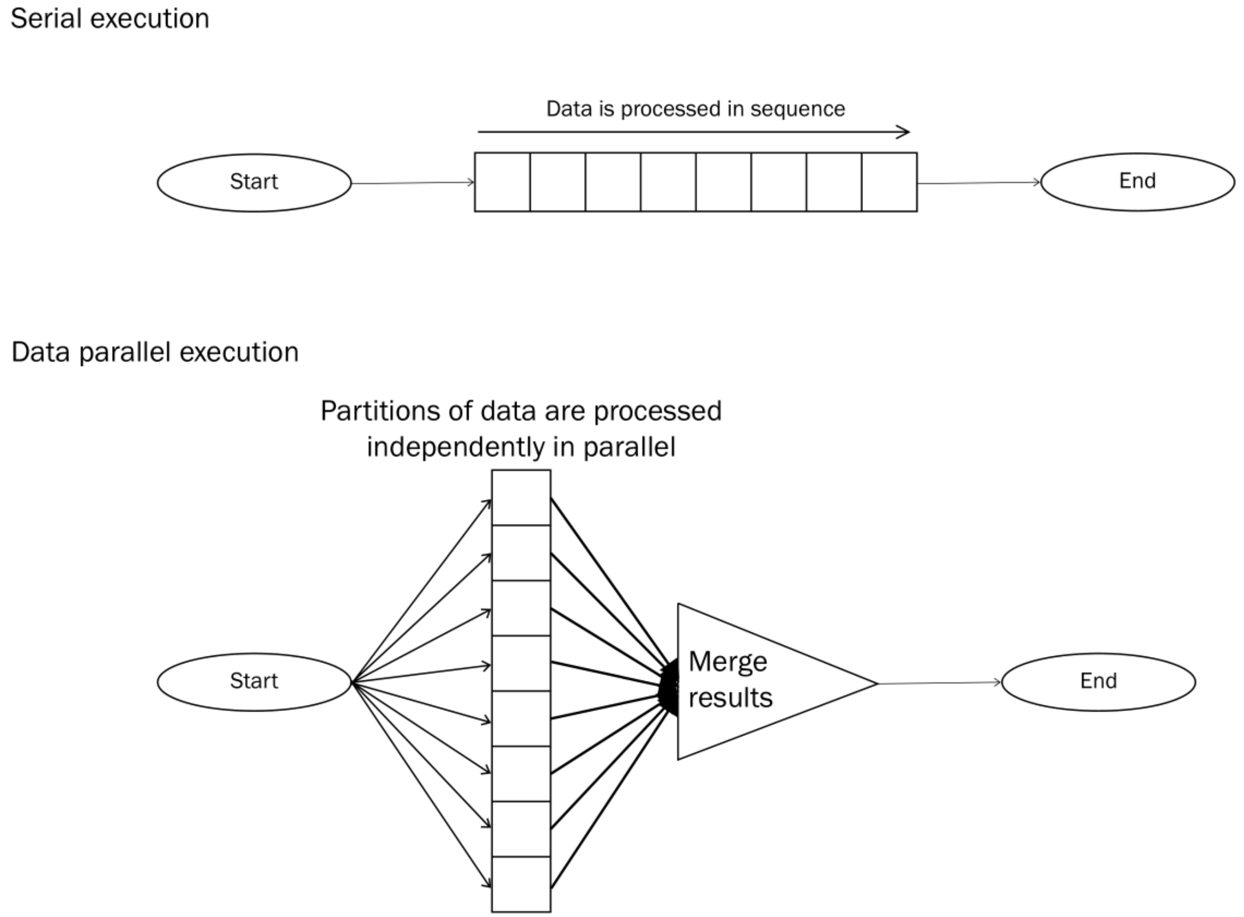
How Much Can Be Parallelized?



https://en.wikipedia.org/wiki/File:AmdahlsLaw.svg

Data Parallelism vs Task Parallelism

- In data parallelism, a dataset is divided into multiple partitions.
- Different partitions are distributed to multiple processors, and the same task is executed on each partition of data.

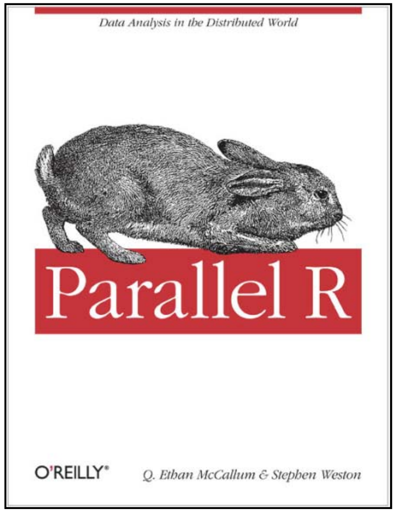


Source: Aloysius Lim, and William Tjhi, R High Performance Programming, Packt Publishing, January 30, 2015.

Data Parallelism vs Task Parallelism

- In task parallelism, tasks are distributed to and executed on different processors in parallel.
  - The tasks on each processor might be the same or different, and the data that they act on might also be the same or different.
  - The key difference from data parallelism: the data is not divided into partitions.
- Example (data parallelism): A random forest is a collection of decision trees built independently on the same data. During the training process for a particular tree, a random subset of the data is chosen as the training set, and the variables to consider at each branch of the tree are also selected randomly. Hence, even though the same data is used, the trees are different from one another.

- **Example (task parallelism)**: Training of a random forest model. In order to train a random forest of say 100 decision trees, the workload could be distributed to a computing cluster with 100 processors, with each processor building one tree. All the processors perform the same task on the same data (or exact copies of the data), but the data is not partitioned.



See *Parallel R* by Q. Ethan McCallum and Stephen Weston.

Get Parallel Results Without Explicitly Doing Parallel Programming

- **`Plyr`** package
  - **`aaply`**: like **`apply`**, but with an option to parallelize.

- **`Foreach`** package
  - Lets you write for loops that can be parallelized (assuming only effect of code is the return value)

- **`Multicore`** package
  - If you have a machine with a multi-core processor, the multicore apply functions (e.g. **`mclapply`**) can farm some set of the data on which the **`apply`** is being called to the other core to speed up the process.
  - Multicore also has functions for more detailed parallel programming, which can make things fast but forces you to confront concurrency (並發性來源) issues.

(IV) Offloading Data Processing to Database Systems

- Extracting data into R versus processing data in a database
  - In some situations, this might not be efficient to manipulate data in R by moving all the data into R whether in memory or on a disk, on a single computer or on a cluster.
  - We can process the data in the database and retrieve only the results into R, which are usually much smaller than the raw data.
  - At the foundation of all in-database tools is the SQL language (relational databases). SQL is a powerful and flexible language used to manipulate data in a database.

- Use an R package as an API to the data warehouse
  - R packages: **`RJDBC`**, **`RODBC`**, **`RPostgreSQL`**, **`RMySQL`**, and **`ROracle`**.

- Converting R expressions to SQL:
  - R packages: **`dplyr`**, **`PivotalR`**, **`MonetDB.R`** and **`SciDB`**.

Example: Running Statistical and Machine Learning Algorithms in a Database

- Open source project, MADlib (http://madlib.net/)
  - brings advanced statistics and machine learning capabilities to PostgreSQL databases, including
  - descriptive statistics, hypothesis tests, array arithmetic, probability functions, dimensionality reduction, linear models, clustering models, association rules, and text analysis.

```
db.drv <- PostgreSQL()
db.conn <- dbConnect(db.drv, host = "hostname",
                     port = 5432, dbname = "rdb",
                     user = "ruser",
                     password = "rpassword)

dbGetQuery(db.conn,
    "SELECT *
    FROM madlib.assoc_rules(
        0.001, -- support
        0.01, -- confidence
        'trans_id', -- tid_col
        'item_id', -- item_col
        'trans_items', -- input_table
        'public', -- output_schema
        TRUE -- verbose
);")
```

Linux (Red Hat), Mac OS X

(VI) Other Consideration

(A) Microsoft R Open

MRAN   About R   Microsoft R Open   Community   Download

Microsoft R Open: The Enhanced R Distribution

Microsoft R Open, formerly known as Revolution R Open (RRO), is **the enhanced distribution of R** from Microsoft Corporation. It is a complete open source platform for statistical analysis and data science.

The current version, Micro
with) R-3.2.5, the most wid
therefore fully compatibili
with that version of R. It in
performance, reproducibi
platforms.

Like R, Microsoft R Open is

Learn more...

Learn More

- About Microsoft R Open
- Reproducibility
- Multithreaded Performance

Microsoft R Application Network

The Microsoft R Portal

Multithreaded Performance

Microsoft R Open includes multi-threaded math libraries to improve the performance of R.

From its inception, R was designed to use only a single thread (processor) at a time. Even today, R works that way unless linked with multi-threaded BLAS/LAPACK libraries.

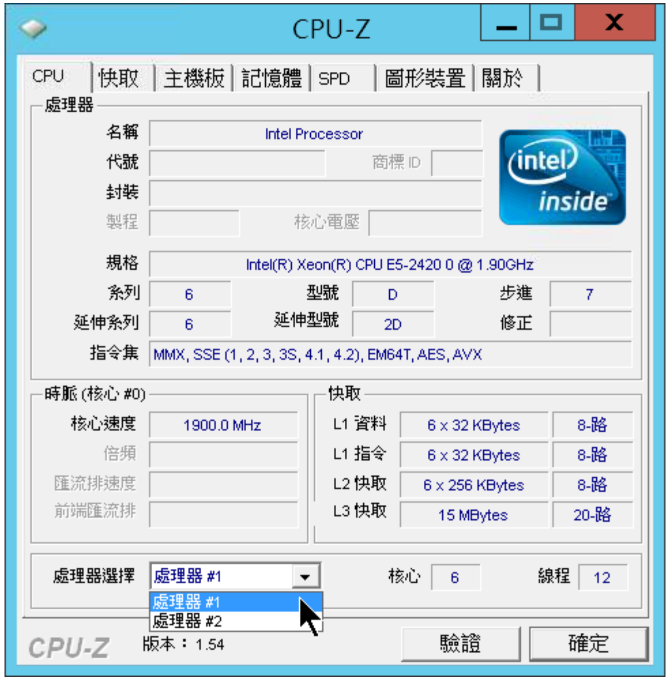
The multi-core machines of today offer parallel processing power. To take advantage of this, Microsoft R Open includes multi-threaded math libraries.
These libraries make it possible for so many common R operations, such as matrix multiply/inverse, matrix decomposition, and some higher-level matrix operations, to compute in parallel and use all of the processing power available to reduce computation times.

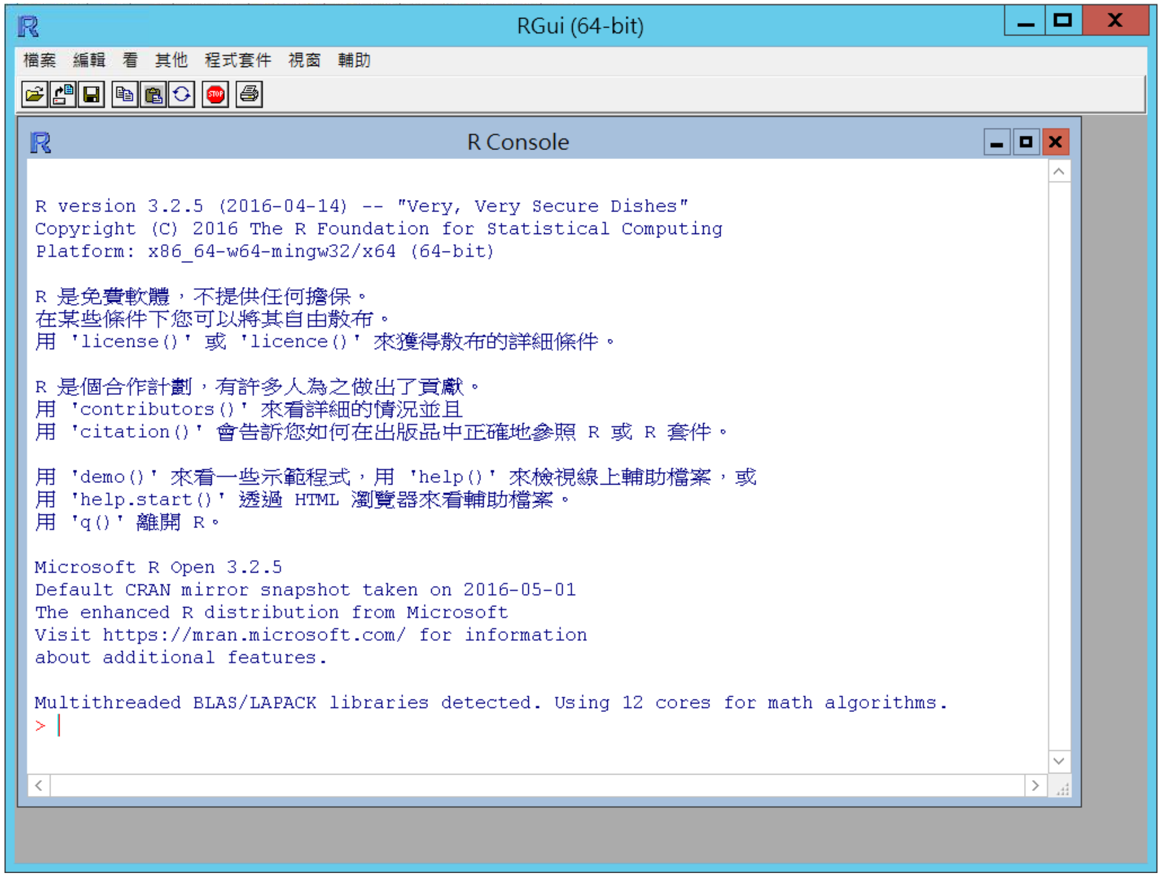
Learn more about how to install and control the number of threads.

See the performance benchmarks.

https://mran.revolutionanalytics.com/documents/rro/multithread/

Multithreading on Windows
Math Kernel Library (MKL)





Performance

Elapsed time（Relative Performance）

	R-3.2.5 (1 thread)		Microsoft R Open-3.2.5 (1 thread)		Microsoft R Open-3.2.5 (4 threads)		Microsoft R Open-3.2.5 (8 threads)	
Matrix multiplication	140.63	1	14.11	9.97	3.27	43.01	1.89	74.41
Cholesky factorization	20.27	1	0.42	48.26	0.35	57.91	0.36	56.31
QR decomposition	4.36	1	2.78	1.57	2.59	1.68	2.63	1.66
Singular value decomposition	52.07	1	1.91	27.26	1.89	27.55	1.90	27.41
Principal component analysis	40.46	1	3.25	12.45	2.67	15.15	2.55	15.87
Linear discriminant analysis	10.14	1	2.36	4.30	2.04	4.97	2.03	5.00

Table Modified from: https://mran.revolutionanalytics.com/documents/rro/multithread/

(B) Programming with Big Data in R

pbdR | About | Packages | Publications

pbdR Packages

- Application
- Communication
- Computation
- Developers
- I/O
- Profiling

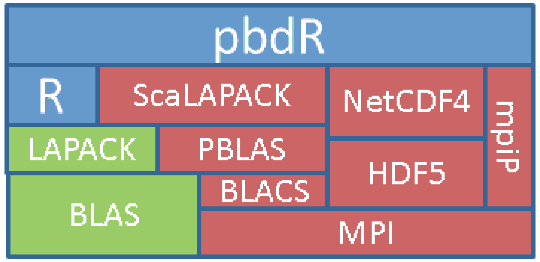
We maintain stable distributions of all of our packages on our GitHub project page. We also try, w
can find current binaries of our packages for the Windows operating system at our sister site, HP

If you are not sure where to begin, start with pbdDEMO package and its vignette. Whether you ar
you up to speed on utilizing the pbdR tools.

Application Specifically purposed packages that utilize pbdR.

Package	Description
pbdML	Machine learning algorithms, using pbdDMAT.
pmclust	Tools for parallel model-based clustering. These include k-means and Gaussian m ad hoc distributed matrices as well as pbdDMAT conformable ones.
pbdDEMO	A collection of pbdR package demonstrations and examples. Also included is a le quickly move R programmers from their laptops to distributed platforms.

http://r-pbd.org/



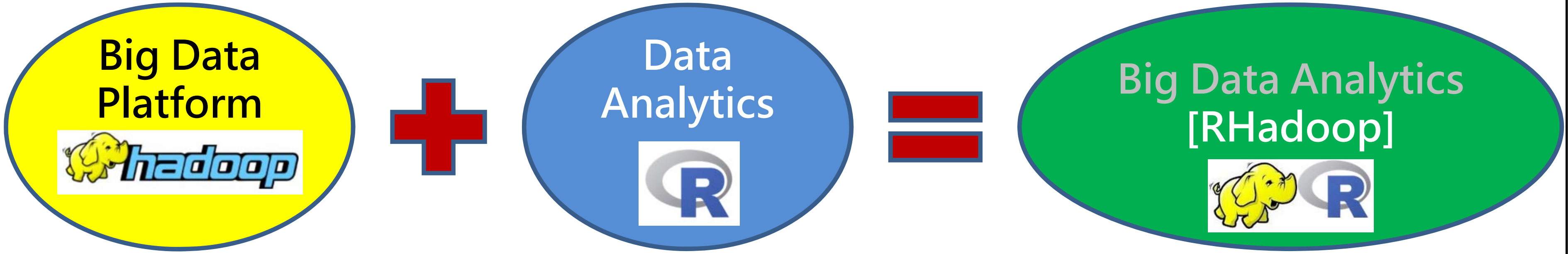
pbdR Relationships to Libraries

a set of R packages for large
uted computing and profiling.
programming
signed to help use R for
ally really big data on high-
computing clusters.

(VI) R and Big Data

- How do analysts use big data:
  - class 1: extract data:
    - extract subset, sample, or summary from a big data source.
    - note that the subset might itself be quite large.
  - class 2: compute on the parts:
    - repeat computations for many subgroups of the data and combine the results.
    - (analogous to map-reduce/split-apply-combine) Can use computing clusters to parallelize analysis.
  - class 3: compute on the whole:
    - the problems of using all of the data at once are irretrievably big.
    - they must be run at scale within the data warehouse.

Hadoop Techniques



- Setup Hadoop Environment (or Using Commercial Services)
  - Linux OS, (or VMware to host Ubuntu within the Windows OS.)
  - Understanding Hadoop features: the Hadoop Distributed File System (HDFS) and MapReduce architecture.
  - Writing Hadoop MapReduce Programs using Java.

- Integrating R and Hadoop:
  - Learning **RHadoop**: performing data analytics with the Hadoop platform via R functions.
  - Learning **RHIPE**: R and Hadoop Integrated Programming Environment (RHIPE) is widely used for performing Big Data analysis with Divide and Recombine (D&R) analysis.
  - Learning **Hadoop streaming:** Hadoop streaming allows you to create and run MapReduce jobs with any executable or R script as the Mapper and/or Reducer. See R package `HadoopStreaming`.

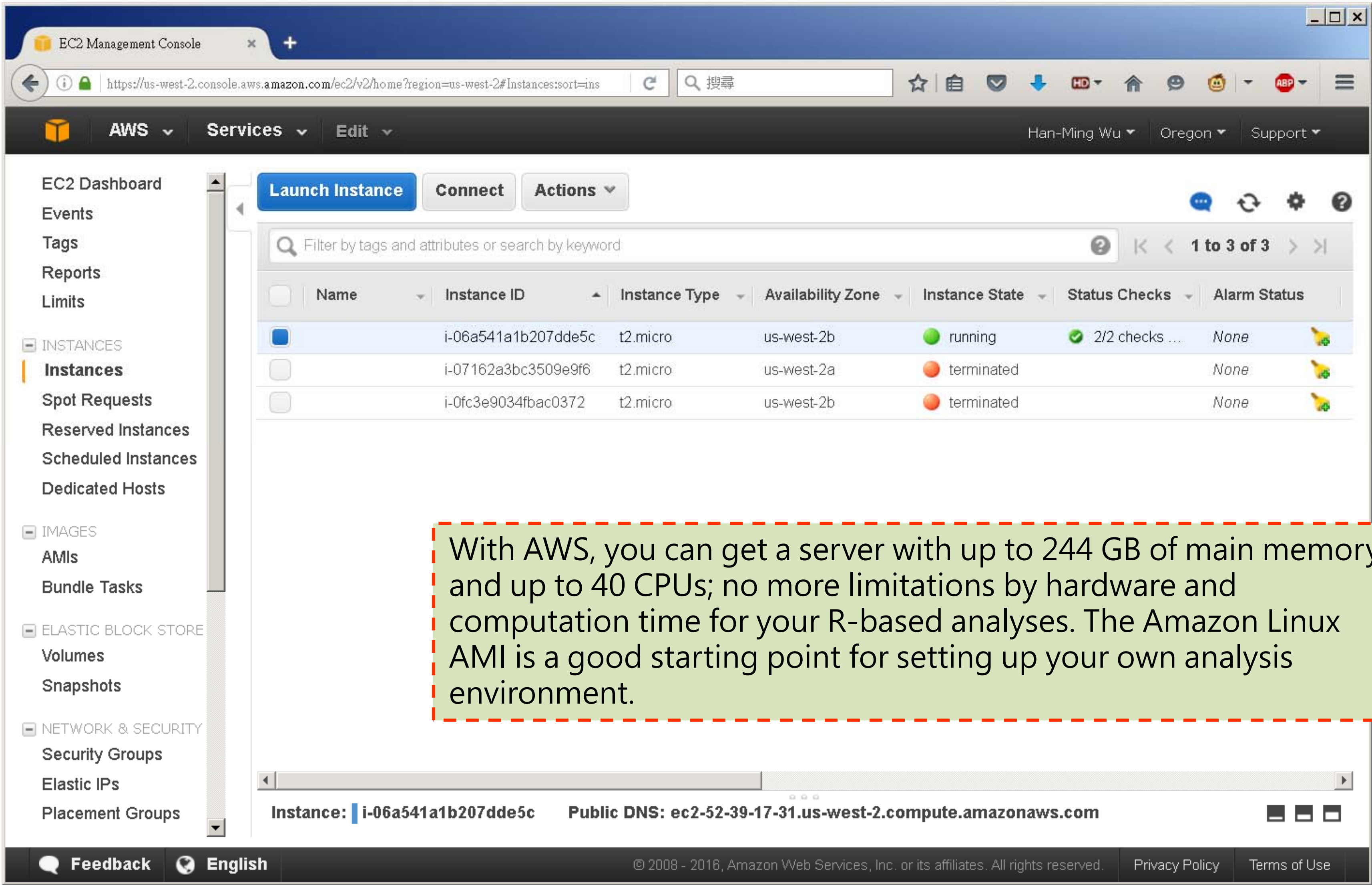
- R packages for importing and exporting data from various DBs: `RMySQL, Xlsx, RMongo, RSQLite, RPostgreSQL, RHive, RHBase, RHDFS`

Amazon Web Services

- m1.xlarge: AWS servers: 4 CPUs and 15 GB of RAM each, with cluster sizes ranging from 4 to 32 core nodes.
- cr1.8xlarge: the largest memory size that an Amazon EC2 node can have is 244 gigabytes.

- Amazon EC2: Amazon Elastic Compute Cloud
- Elastic MapReduce (EMR) allows to rent and run a Hadoop cluster on an hourly basis.
- An Amazon Machine Image (AMI) defines which operating system is installed on your EC2 instance and which software is included.

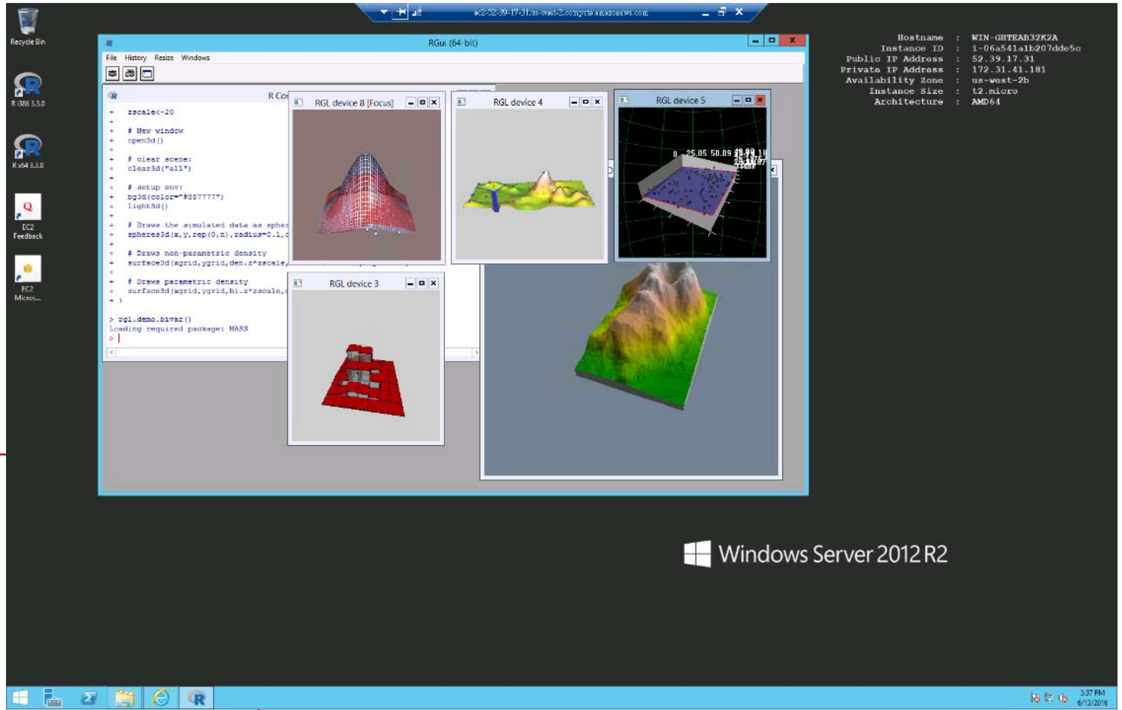
Amazon Web Services



With AWS, you can get a server with up to 244 GB of main memory and up to 40 CPUs; no more limitations by hardware and computation time for your R-based analyses. The Amazon Linux AMI is a good starting point for setting up your own analysis environment.

Setting Up an AWS Instance for R

- Uploading data to HDFS
- Analyzing HDFS data with RHadoop
- R code for MapReduce

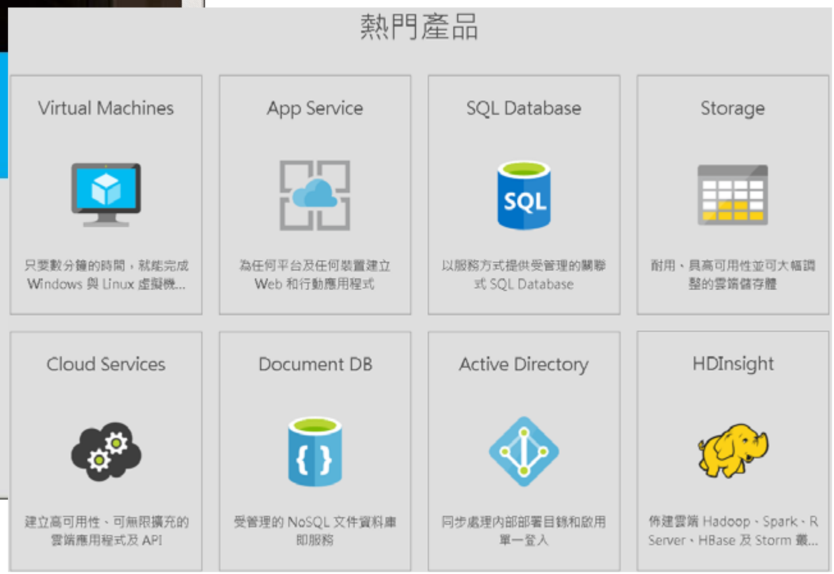



```
mapper <- function(keys, values) {
    ...
}

reducer <- function(key, values) {
    ...
}

job <- mapreduce(input = "/data/myinput",
                 input.format = input.format,
                 output = "/data/myoutput",
                 map = mapper, reduce = reducer)

results <- from.dfs(job)
```

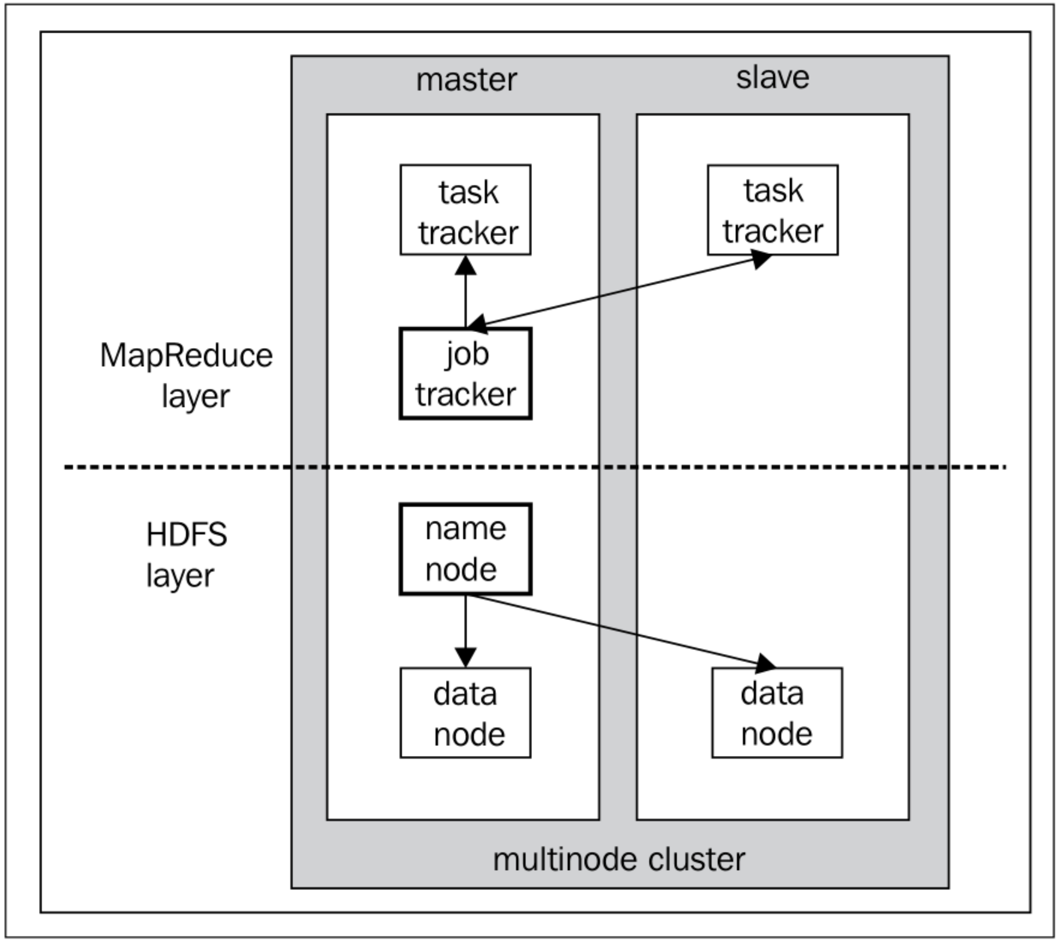
How to set up an AWS instance for R

- https://blogs.aws.amazon.com/bigdata/post/Tx3IJSB6BMHWZE5/Running-R-on-AWS
- http://www.r-bloggers.com/setting-up-an-aws-instance-for-r-rstudio-opencpu-or-shiny-server/
- https://www.r-project.org/conferences/useR-2010/tutorials/Zolot_tut.pdf

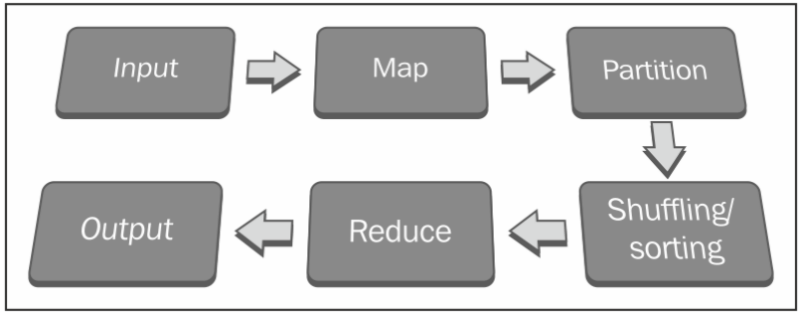
Microsoft Azure



The Four Main Stages of Hadoop MapReduce Data Processing



The HDFS and MapReduce architecture



- The loading of data into HDFS
- The execution of the Map phase
- Shuffling and sorting
- The execution of the Reduce phase

- Map: 從主節點(master node)輸入一組input，此input是一組key/value，將這組輸入切分成好幾個小的子部分，分散到各個工作節點(worker nodes)去做運算
- Reduce: 主節點(master node)收回處理完的子部分，將子部分重新組合產生輸出

Example: The Steps to Run a MapReduce Job with Hadoop

`Map.java`: the Map class for the word count Mapper.

```
// Defining package of the class
package com.PACKT.chapter1;

// Importing java libraries
import java.io.*;
importjava.util.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

// Defining the Map class
public class Map extends MapReduceBa
         Mapper<LongWritable,
                Text,
                Text,
                IntWritable>{

//Defining the map method - for proc
problem specific logic
public void map(LongWritable key,
                Text value,
                OutputCollector<Text

                   ...

// Emitting the (key,value) pair wit
output.collect(new Text(st.nextToken
               new IntWritable(1));
         }
```

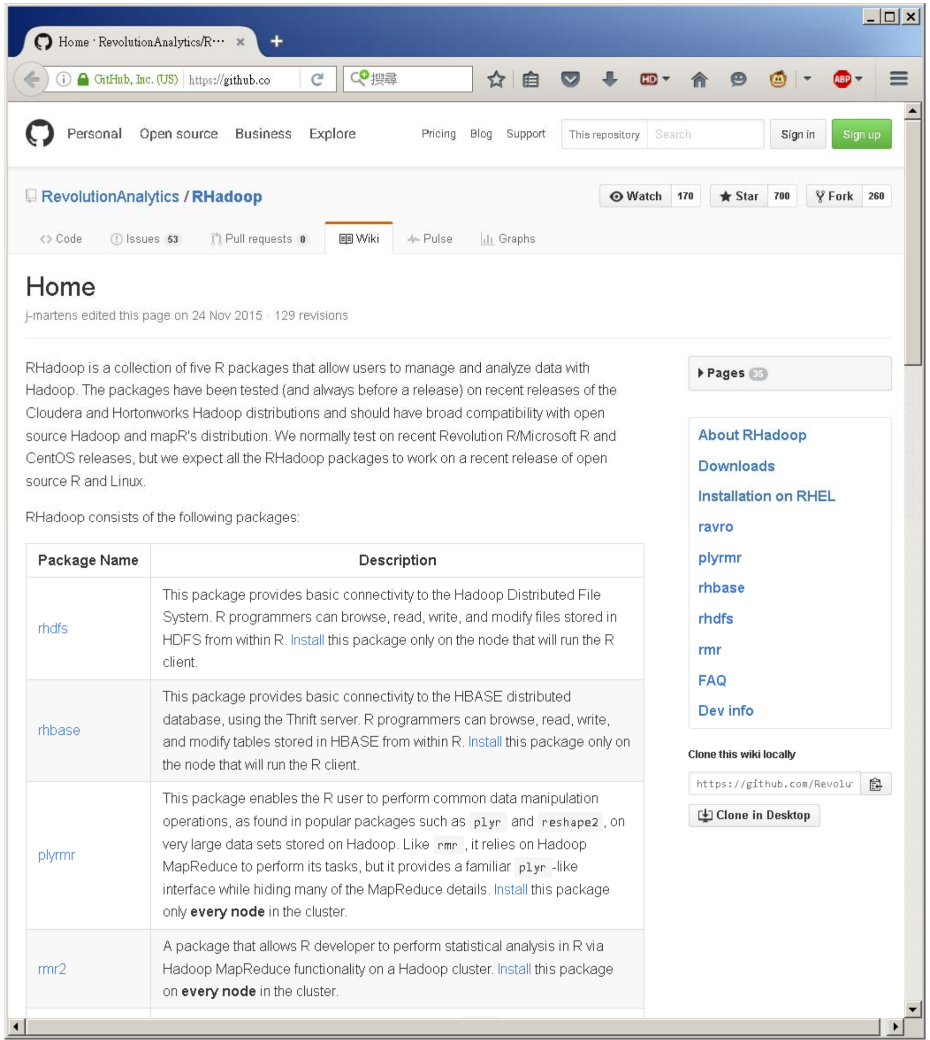
`Reduce.java`: the Reduce class for the word count Reducer.

```
// Defining package of the class
package com.PACKT.chapter1;

// Importing java libraries
import java.io.*;
importjava.util.*;
import org.apache.hadoop.io.*;
importorg.apache.hadoop.mapred

// Defining the Reduce class
public class Reduce extends Ma
          Reducer<Text,
                  IntWritable,
                  Text,
                  IntWritable>

// Defining the reduce method
generated output of Map phase
public void reduce(Text key,
                   Iterator<In
                   OutputColle
                   output,
                   Reporter re

                       ...

// Emitting the (key,value) pa
output.collect(key, new IntWri
    }
}
```

`WordCount.java`: the task of Driver in the Hadoop MapReduce Driver main file.

```
//Defining package of the class
package com.PACKT.chapter1;

// Importing java libraries
import java.io.*;
importorg.apache.hadoop.fs.*;
import org.apache.hadoop.io.*;
importorg.apache.hadoop.mapred.*;
importorg.apache.hadoop.util.*;
importorg.apache.hadoop.conf.*;

//Defining wordcount class for job configuration
  // information
public class WordCount extends Configured implements Tool{

                         ...

// For submitting the configuration object
JobClient.runJob(conf);

return 0;
    }

// Defining the main() method to start the execution of //
the MapReduce program
public static void main(String[] args) throws Exception {
  intexitCode = ToolRunner.run(new WordCount(), args);
  System.exit(exitCode); } }
```

RHadoop


Home · RevolutionAnalytics/R···

RevolutionAnalytics / RHadoop

Watch 170 | Star 700 | Fork 260

Code | Issues 53 | Pull requests 0 | Wiki | Pulse | Graphs

Home

j-martens edited this page on 24 Nov 2015 · 129 revisions

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.

RHadoop consists of the following packages:

Package Name	Description
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.

Pages 35

About RHadoop
Downloads
Installation on RHEL
ravro
plyrmr
rhbase
rhdfs
rmr
FAQ
Dev info

Clone this wiki locally

Clone in Desktop

See Also **mapReduce** package.

```r
map <- function(k,lines) {
    words.list <- strsplit(lines, '\\s')
    words <- unlist(words.list)
    return( keyval(words, 1) )
}

reduce <- function(word, counts) {
    keyval(word, sum(counts))
}

wordcount <- function (input, output=NULL) {
    mapreduce(input=input, output=output,
        input.format="text", map=map,
        reduce=reduce)
}
## read text files from folder wordcount/data
## save result in folder wordcount/out
## Submit job

hdfs.root <- 'wordcount'
hdfs.data <- file.path(hdfs.root, 'data')
hdfs.out <- file.path(hdfs.root, 'out')
out <- wordcount(hdfs.data, hdfs.out)

## Fetch results from HDFS
results <- from.dfs(out)
results.df <- as.data.frame(results,
    stringsAsFactors=F)
colnames(results.df) <- c('word', 'count')
head(results.df)
```

Source http://www.rdatamining.com/big-data/rhadoop

MapReduce in R

Logistic Regression in R

```
model <- glm(response ~., family=binomial(link='logit'), data=mydata)
```

Using Mapreduce Framework in R

```
lr.map <- function(., M){
    Y <- M[,1]
    X <- M[,-1]
    keyval(1, Y * X * g(-Y *
        as.numeric(X %*% t(plane))))
}

lr.reduce <- function(k, Z){
     keyval(k, t(as.matrix(
        apply(Z,2,sum))))
}
```

```
logistic.regression <- function(input,
iterations, dims, alpha){

    plane = t(rep(0, dims))
    g <- function(z) 1/(1 + exp(-z))
    for(i in 1:iterations){
        gradient <- values(from.dfs(
            mapreduce(input,
                map = lr.map,
                reduce = lr.reduce,
                combine = T)))
        plane = plane + alpha * gradient
    }
    plane
}
```

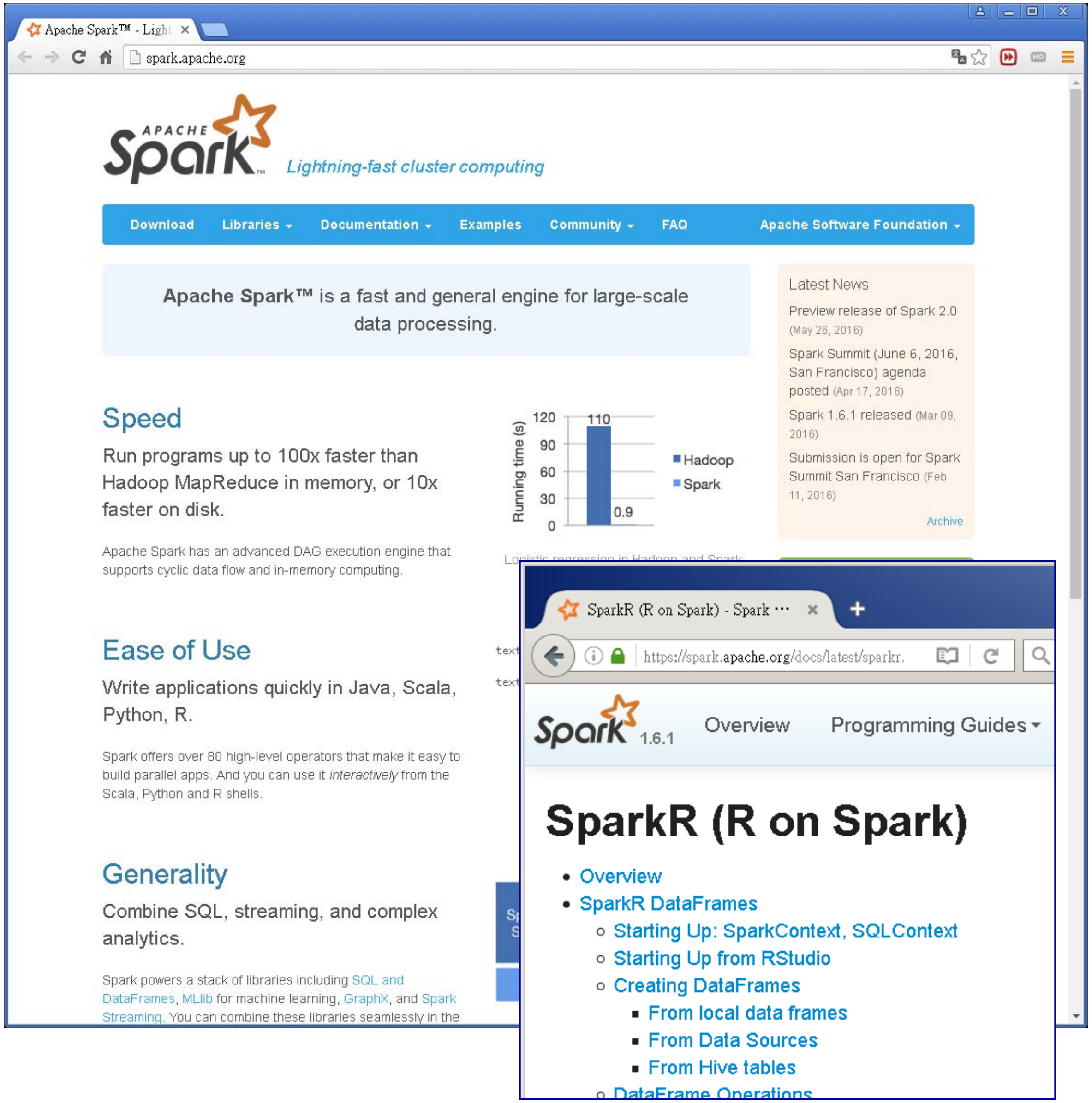
logistic regression by gradient descent

Source: https://github.com/RevolutionAnalytics/rmr2/blob/master/docs/tutorial.md

References

- Large Datasets and You:
  - http://www.mattblackwell.org/files/papers/bigdata.pdf
- FasteR! HigheR! StrongeR! - A Guide to Speeding Up R Code for Busy People:
  - http://www.noamross.net/blog/2013/4/25/faster-talk.html
- Scalable Strategies for Computing with Massive Data:
  - http://www.slideshare.net/fullscreen/joshpaulson/big-memory/1
- Taking R to the Limit, Part II: Working with Large Datasets:
  - http://www.bytemining.com/wp-content/uploads/2010/08/r_hpc_II.pdf
  - https://github.com/rstudio/webinars/blob/master/14-Work-with-big-data/14-Work-with-big-data.pdf
- 20140317 MLDM Monday - 使用 RHadoop 做巨量資料分析
  - https://www.youtube.com/watch?v=vmIHge8JSXg
- MapReduce and R: Short example on regression / forecasting
  - https://www.youtube.com/watch?v=Q8kmAfpwAJQ
- RDataMining.com: R and Data Mining, Building an R Hadoop System
  - http://www.rdatamining.com/big-data/rhadoop
- MapReduce in R
  - https://github.com/RevolutionAnalytics/rmr2/blob/master/docs/tutorial.md
- Using R with Hadoop
  - http://www.revolutionanalytics.com/free-webinars/using-r-hadoop

Apache Spark™
Lightning-Fast Cluster Computing



技術準備

- 不怕接觸 Linux (Ubuntu/ Centos)
- 有程式設計開發經驗(C#, Java, Python, SQL, R)
- 基本Java或Scala的概念
- 熟悉 SQL 語法與應用。

Apache Spark provides Resilient Distributed Datasets (RDDs) that store data in memory across a Hadoop cluster. This allows data to be read from HDFS once and to be used many times in order to dramatically improve the performance of interactive tasks like data exploration and iterative algorithms like gradient descent or k-means clustering.
Spark在記憶體內執行程式的運算速度能做到比Hadoop MapReduce的運算速度快上100倍，即便是執行程式於硬碟時，Spark也能快上10倍速度。
[https://zh.wikipedia.org/wiki/Apache_Spark]

結論

- 要學的東西太多，但時間太少。
- 「程式可以跑就好啦! 幹嘛要寫的好看、寫的有效率啊?」
  - 心態問題:
    事情有做就好 vs 把事情做好。
- 統計(角色、任務等等)在哪裡?
- 「如果你本來就認為數據有大用那剩下的就是技術問題；如果你本來就認為數據無大用，那剩下的都是問題」*(精誠集團Etu負責人蔣居裕)*

講義下載
http://www.hmwu.idv.tw/

「語文永遠不是阻止我們了解事物的最大障礙，求知欲才是!」

練習1: 不同R函式計算n!所需的時間

```
factorial.for <- function(n){
    f <- 1
    if(n<2) return(1)
    for(i in 2:n){
        f <- f*i
    }
    f
}
factorial.for(5)
```

```
factorial.while <- function(n){
    f <- 1
    t <- n
    while(t>1){
        f <- f*t
            t <- t-1
    }
    return(f)
}
factorial.while(5)
```

```
factroial.repeat <- function(n){
    f <- 1
    t <- n
    repeat{
        if(t<2) break
            f <- f*t
        t <- t-1
    }
    return(f)
}
factroial.repeat(5)
```

```
factorial.call <- function(n, f){

    if(n <= 1){
      return(f)
    }
    else{
      factorial.call(n-1, n*f)
    }
}
factorial.call(5, 1)
```

```
factorial.cumprod <- function(n){
    max(cumprod(1:n))
}
factorial.cumprod(5)
```

```
my.factorial <- function(n){
     ...
}
Rprof("output.txt")
ans <- my.factorial(5000)
Rprof(NULL)
summaryRprof("output.txt")
```

練習2

- 僅輸入所需要的部份資料，而不是全部。

```
Variables <- c("NULL", "NULL", "factor", "numeric")
myData <- read.table("fileName", colClasses = Variables)
```

- 用適合的函式或演算法: O(N) vs O(N^2)

```
x <- 1:10000; s <- sample(x, 10)
a1 <- which(x %in% s)
a2 <- intersect(x, s)
a3 <- which(is.element(x, s))

for(i in 1:10000){
    for(j in 1:10){
        if(all.equal(x[i], s[j])){
            ...
        }
    }
}
```

```
> n <- 10000
> p <- 1000
> Mat <- matrix(rnorm(n*p),
nrow=n, ncol=p)
> system.time(apply(Mat, 1, sum))
   user  system elapsed
   0.61    0.19    2.56
> system.time(rowSums(Mat))
   user  system elapsed
   0.05    0.00    0.08
```

練習3

- 資料儲存以二進位檔(binary)為優先:
  - 讀寫文字檔比壓縮二進位檔慢。
  - 壓縮二進位檔又比二進位慢。

```
> n <- 1000
> p <- 1000
> Mat <- matrix(rnorm(n*p),
nrow=n, ncol=p)
```

```
> system.time(write.table(Mat, file="myData.txt"))
   user  system elapsed
   8.89    0.09   12.14
> system.time(read.table("myData.txt"))
   user  system elapsed
  10.85    0.06   11.98
```

```
> system.time(save(Mat, file="myData.gz"))
   user  system elapsed
   1.11    0.01    2.52
> system.time(load("myData.gz"))
   user  system elapsed
   0.36    0.02    3.56
```

```
> system.time(save(Mat, file="myData.Rdata", compress=FALSE))
   user  system elapsed
   0.24    0.00    0.23
> system.time(load("myData.Rdata"))
   user  system elapsed
   0.23    0.00    0.24
```

練習4

- 初始向量變數時就先給定長度。避免大量回圈(loop)。
- 採用向量化的運算方式。使用Rprof()檢查程式區塊。

```
comp1 <- function(n){
    x <- numeric()
    for(i in 1:n){
        x[i] <- log(i);
    }
}
```

```
comp2 <- function(n){
    x <- numeric(n)
    for(i in 1:n){
        x[i] <- log(i);
    }
}
```

```
comp3 <- function(n){
    x <- numeric(n)
    for(i in 1:n) {
        x[i] <- log(i);
    }
}
```

```
comp4 <- function(n){
    i <- 1:n
    x <- log(i)
}
```

```
mycomp <- function(n){
    comp1(n)
    comp2(n)
    comp3(n)
    comp4(n)
}
```

```
> Rprof()
> mycomp(10000)
> Rprof(NULL)
> print(summaryRprof())
$by.self
       self.time self.pct total.time total.pct
comp1       0.46     79.3       0.46      79.3
comp2       0.06     10.3       0.06      10.3
comp3       0.06     10.3       0.06      10.3
mycomp      0.00      0.0       0.58     100.0

$by.total
       total.time total.pct self.time self.pct
mycomp       0.58     100.0      0.00      0.0
comp1        0.46      79.3      0.46     79.3
comp2        0.06      10.3      0.06     10.3
comp3        0.06      10.3      0.06     10.3

$sampling.time
[1] 0.58
```