Programación Paralela con R y Rstudio.
Una Introducción

Instructor Rina Surós

Noviembre 2022

Facultad de Ingeniería. UDELAR

Montevideo

Como tratar datos que superan por mucha la capacidad de la memoria RAM

- Algoritmos Paralelos y Big Data
- **package ff**

Algoritmos Paralelos y Big Data

Ahora nos ocuparemos de problemas realmente complejos que requieren combinar Algoritmos Paralelos y Big Data

***Big Data, macrodatos, datos masivos**, son términos que escuchamos comúnmente, el término fue adjudicado en el año 2001.*

- *Quienes son los más interesados en la actualidad: las empresas. Estas se dieron cuenta del valor que puede tener para ellas el análisis de volúmenes masivos de datos y sus consecuencias económicas.*

- *Los científicos e investigadores estaban más acostumbrados a lidiar con problemas con un gran volumen de datos, muy difíciles de manejar.*

Algoritmos Paralelos y Big Data

Los científicos usan, como hemos visto en este curso, técnicas de ***observación de datos por ventanas de tiempo****.*

Pero la velocidad de generación de los datos no permite pensar en análisis en tiempo real, que es una necesidad.

Ejemplo*: Un sistema de monitoreo y control de producción SCADA es multidimensional (hasta 200.000 variables). Y el análisis se requiere en tiempo real*

Los sistemas SCADA ofrecen una interfaz gráfica PC-Operario tipo HMI -para la supervisión- que pueden monitorear procesos.

Características principales de un SCADA :

- Adquisición y almacenado de datos para recoger, procesar y almacenar la información recibida en forma continua y confiable.
- Representación gráfica y dinámica de variables de proceso y su monitorización por medio de alarmas
- Ejecutar acciones de control para modificar la evolución del proceso, actuando ya sea sobre los reguladores autónomos básicos (consignas, alarmas, menús, etc.) o directamente sobre el proceso mediante las salidas conectadas.
- Arquitectura abierta y flexible con capacidad de ampliación y adaptación.

- Conectividad con otras aplicaciones y bases de datos, locales o distribuidas en redes de comunicación.
- Supervisión, para observar desde un monitor la evolución de las variables de control.
- Transmisión de información con dispositivos de campo y otros PC.
- Base de datos, gestión de datos con bajos tiempos de acceso.
- Presentación, representación gráfica de los datos. Interfaz del Operador.
- Explotación de los datos adquiridos para gestión de la calidad, control estadístico, gestión de la producción y gestión administrativa y financiera.
- Alertar al operador sobre cambios detectados en la planta, tanto aquellos que no se consideren normales (alarmas) como los que se produzcan en su operación diaria (eventos). Estos cambios son almacenados en el sistema para su posterior análisis.

Las prestaciones que ofrece un sistema SCADA eran impensables hace una década y son las siguientes:

- Posibilidad de crear paneles de alarma, que exigen la presencia del ordenador para reconocer una parada o situación de alarma, con registro de incidencias.
- Generación de datos históricos de señal de planta, que pueden ser incorporados para su proceso sobre una hoja de cálculo.
- Creación de informes, avisos y documentación en general.
- Posibilidad de programación numérica, que permite realizar cálculos aritméticos de elevada resolución sobre la CPU del ordenador y no sobre la del autómata, menos especializado, etc.

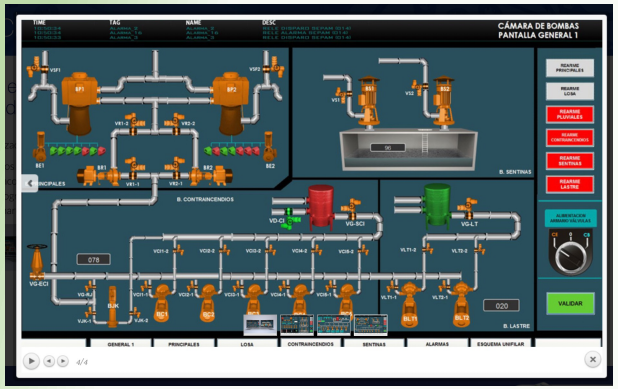
Con estas herramientas, se pueden desarrollar aplicaciones basadas en el PC, con captura de datos, análisis de señales, presentaciones en pantalla, envío de resultados a disco o impresora, control de actuadores, etc. (Gómez et al., 2008).

¡¡¡Y todo eso en TIEMPO REAL!!!

¿Es posible generar información en tiempo real sobre hasta 200.000 variables, conociendo las limitaciones de R?

Analizar datos estadísticamente, generar gráficas estáticas -y mejor aún- dinámicas y analizarlas, tomar decisiones sobre la marcha, etc; todas esatas son necesidades actuales de las empresas. Por eso los analistas de datos necesitan manejar herramientas que aceleren el proceso de análisis.

Y muchísimas cosas más



TAREAS QUE SE PUEDEN AUTOMATIZAR CON SISTEMAS SCADA

Un SCADA (control de supervisión y adquisición de datos) es un sistema de control de automatización que se utiliza en industrias tales como energía, petróleo y gas, agua, energía y muchas más.

Ejemplo de Carta Dinagráfica

CLASE	NOMBRE DE FALLA	Nº
1	Bomba normal	
2	Bomba desgastada	
3	Pistón golpeando arriba de la bomba	
4	Válvula fija mala	
5	Válvula viajera mala	
6	Bomba mala	
7	Pistón golpeando por abajo	
8	Falla en barril	
9	Suciedad de la bomba	
10	Golpe de fluido	
1	Tubería sin anclar	
12	Interferencia de gas	
13	Golpe de fluido con fricción y movimiento de tubería	
14	Pistón pegado sin sobrepasar el limite elástico de las ca	
15	Agujero en el cilindro de la bomba	
16	Cabillas partidas	

Interfaz gráfica dinámica

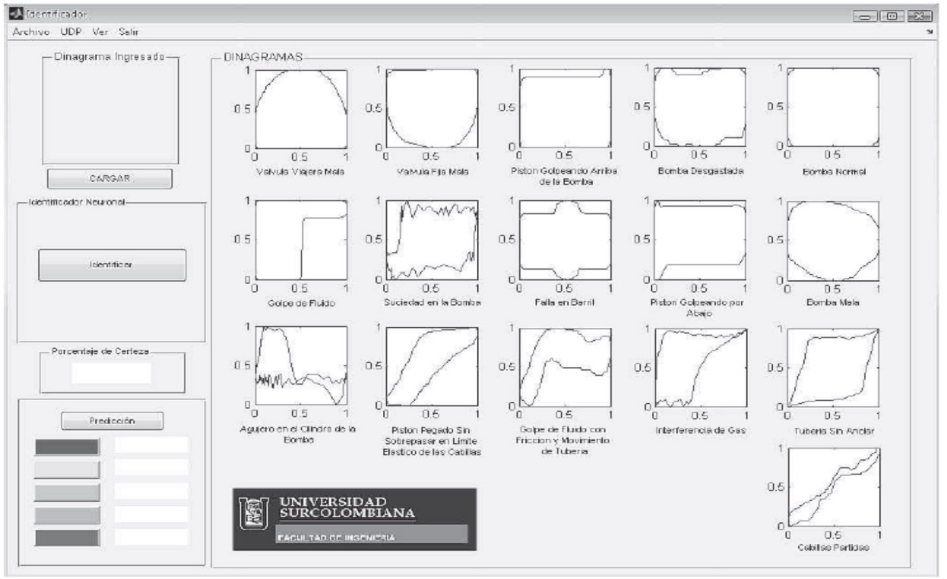


Figura 5. Pantalla principal de Dinasoft

Una red neuronal entrenada para identificar fallas asociadas con el dinagrama de fondo de un **sistema de bombeo mecánico** (en un pozo petrolero), posee una interfaz gráfica en donde el analista u operario puede visualizar todaslas diferentes formas de cartas dinagráficas que se presentan en un campo particular con múltiples pozos.

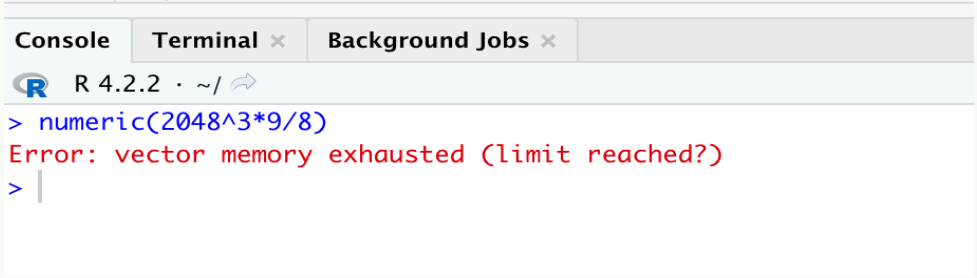
 # En un computador Mac OS X, con 8 Gb de RAM, con 64-Bit

Vamos en Rstudio el cálculo, *cuando se trabaja con datos*

 # R no es capaz de asignar el vector

*numeric(2048^3*9/8)*

Cuando se trabaja con datos relativamente grandes en RStudio, es posible encontrar un error:



¿Es la RAM? ¿es Rstudio? ¿ambos?

Tratar grandes cantidades de datos y además *acelerar los procesos*

Ahora nos ocuparemos de problemas que combinan dos características

- Grandes cantidades de datos. Siempre que los necesitemos todos
- Necesidad de acelerar procesos. Siempre que el tiempo de procesamiento sea importante

Nos centraremos en resolver problemas en nuestro computador personal, así que veamos el siguiente paquete que permite gestionar la memoria en el caso en que necesitemos tratar Big Data

package ff

Y usemos además un paquete paralelo

Tratar grandes cantidades de datos y además *acelerar los procesos*

El paquete **“ff”** se utiliza en los casos en que tenemos que procesar, de forma simultánea, una cantidad de datos superior a la capacidad de nuestra memoria RAM. Eso significa en mi caso 8 Gb de datos tratados simultáneamente.

Pero si podemos almacenar esos datos en la memoria RAM

Memoria RAM <= datos < DD

- **No podemos utilizar la mayoría de las funciones clásicas de R, no funcionan. Cada vez que quiera aplicar una función conocida, tendré que buscar el equivalente en “ff”**

- Tendremos una nueva familia de funciones, veremos algunas de ellas.
- Este paquete permite usar la información en DD (bases de datos u objetos) como si usáramos los métodos comunes, los más convencionales
- En el DD el almacenamiento se hacer por particiones

Paquete ff

El paquete **ff** está asociado con otros paquetes de modelización, como por ejemplo: Biglm para modelos lineales

https://rubenfcasal.github.io/intror/modelado-de-datos.html

Este paquete es de mucha utilidad en técnicas de *machine learning*, que requieren utilizar grandes cantidades de datos simultáneos, imposibles de procesar. En el área de aprendizaje de máquinas

Para tratar con varias bases de datos relacionadas entre si

Los códigos se simplifican enormemente.

Paquete ff con bases de dato estructuradas

Si la base de datos es **estática (invariables en el tiempo)**:

- El preprocesamiento y limpieza de los datos se hace una sola vez
- La gestión de memoria es la misma para todo el proceso.

Si la base de datos es **dinámica (Variable en el tiempo)**: análisis de datos en tiempo real. Ej: sistemas skada

- Si la BD cambia constantemente, tendremos que hacer el preprocesamiento y limpieza en casa nuevo ciclo
- La gestión de memoria es dinámica, cambia para cada instante de tiempo. “Tratamos estados de la base de datos” en el tiempo

Paquete ff

Cargar el Package 'ff'

Usaremos las librerías:

library(ff) # manejo transparente de grandes volúmenes de datos
#library(ffbase) # funciones estadísticas básicas
library(readr) # para lectura de datos csv, tsv, fwf.
library(snowfall) # para el desarrollo de aplicaciones paralelas

 # Usaremos el archivo de datos: 2008, "Data Expo 2009: Airline on time data".

Este archivo Comprimido pesa 1.5 Gb. Se encuentra en:

https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/HG7NV7

Ejemplos con *aerolinea_2008.csv*

El archivo aerolinea_2008.csv es solo el año 2008 al que se le dejaron un poco más de 500 mil líneas.

La versión utilizada en el script: aerolínea_2008.r no es la original, se eliminaron algunas columnas por comodidad.

 # Para los cálculos estadísticos usaremos "resumen" que arroja: mínimo, media, máximo y largo de un vector

resumen = function(x) c(min = min(x), media = mean(x), max=max(x),largo=length(x))

*# Para la lectura de datos con **ff** se usa:*

***read.csv.ffdf(**file="2008V.csv",VERBOSE = TRUE,first.rows = -1,header=T)*

 # Usaremos gc() para administrar de forma automática la memoria, liberar los objetos que ya no están en uso y que no serán usados en el futuro (temporales).

***gc**(verbose = getOption("verbose"), reset = FALSE, full = TRUE)*
gcinfo(verbose)

Veamos la ocupación de la memoria cuando hacemos la "lectura ff"

 # Lectura comun

```
aero1 <- read_csv("2008V.csv")
dim(aero1) # dimension del arreglo filas x columnas
format(object.size(aero1),"Mb") # megas que ocupa de memoria el arreglo
```

#Resultado: "829.2 Mb"

 # Lectura con ff

```
aero2=read.csv.ffdf(file="2008V.csv",VERBOSE = TRUE,first.rows = -       1,header=T)
dim(aero2) # dimension del arreglo
format(object.size(aero2),"Mb") # megas que ocupa de memoria el arreglo
```

 # Resultado: "0.1 Mb"

¡¡¡Impresionante!!!

```
> library(ff)
> library(readr)
> library(snowfall)
> library(snow)
> resumen = function(x) c(min = min(x), media = mean(x), max=max(x),largo=length(x))
> resumen(c(1,5,3)) #ejemplo de uso
  min media   max largo
    1     3     5     3
> #########
> # Tamaño del archivo csv de interes
> file.size("aerolineas.csv")/1024/1024
[1] 358.3962
> start<-Sys.time()
>
> aero1 <- read_csv("aerolineas.csv")
Rows: 21735733 Columns: 5
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (3): Origin, Dest, CancellationCode
dbl (2): ArrDelay, Distance

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
> dim(aero1) # dimension del arreglo: número de líneas
[1] 21735733        5
> format(object.size(aero1),"Mb") # megas que ocupa de memoria el arreglo
[1] "829.2 Mb"
>
> object.size(aero1)
869470344 bytes
> print(Sys.time() - start)
Time difference of 10.8514 secs
>
```

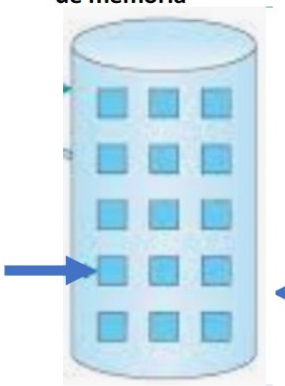
```
> # Lectura con ff
>
> start<-Sys.time()
> aero2=read.csv.ffdf(file="aerolineas.csv",VERBOSE = TRUE,first.rows = -1,header=T)
read.table.ffdf 1..21735733 (21735733)  csv-read=13.572sec ffdf-write=1.185sec
 csv-read=13.572sec  ffdf-write=1.185sec  TOTAL=14.757sec
> dim(aero2) # dimension del arreglo
[1] 21735733        5
> format(object.size(aero2),"Mb") # megas que ocupa de memoria el arreglo
[1] "0.1 Mb"
> object.size(aero2)
61688 bytes
>
> print(Sys.time() - start)
Time difference of 14.76024 secs
>
>
> # Uso de memoria
> # memory.size (max = F) # para windows
> # memory.limit()        # para windows
> # Informa sobre la memoria y ayuda a limpiar
> object.size(aero2)
61688 bytes
> # este archivo está en disco
> gc()
            used   (Mb) gc trigger   (Mb) limit (Mb)  max used   (Mb)
Ncells  15253076  814.7   23720347 1266.9         NA  23720347 1266.9
Vcells 142904754 1090.3  448310417 3420.4      16384 448160631 3419.2
>
>
>
```

Sistema de paginación de memoria

Pero **0,1 Mb** corresponde a la memoria virtual. Al arreglo que contiene únicamente las direcciones de memoria, **metadata**. Cuando se invocan datos en memoria, la RAM a su vez hace una invocación a la dirección de la memoria del DD (o memoria externa) donde se encuentran esos datos y los trae para procesar

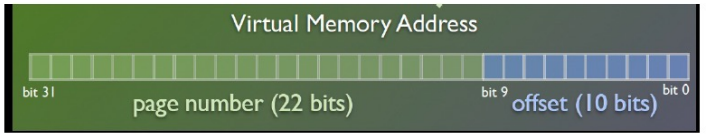
- **Los datos reciden en el DD**
- **Hay un sistema de paginación de memoria**

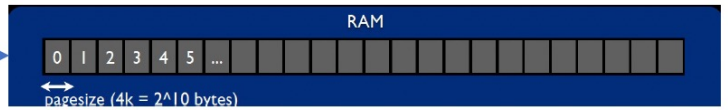
Página de memoria



Swap In/Out

Arreglo que contiene las direcciones de memoria que hace posible cargar los datos desde el DD a la memoria RAM





Sistema de paginación de memoria

El sistema de paginación de memoria, en la practica, enlentece “un poco” el proceso. Pero ganamos en capacidad de datos a tratar y los códigos son más simples.

Además, el sistema operativo utiliza la memoria “cache” para traer del DD la información a ser tratada inmediatamente.

En el DD en general tengo mucho espacio disponible

Puedo agregar DD externos

Todo esto conforma una enorme potencialidad de cálculo

Vamos al script: aerolinea_2008.r

Otras características del uso de “ff”

 # En secuencial

chunk (bloque) se usa con objetos que provengan de **"ff”,** es una función que permite tratar la información por trozos (particiones) que la función selecciona automaticamente.

indicamos el tamaño de las particiones, con la columna distancia

calcular en 10 paquetes (10 particiones). Usando función "resumen”

```
p1 <- lapply( chunk(dist,length = 10) , function(i) resumen(dist[i]))
```

 # En paralelo

*# chunk se usa con objetos que provengan de **"ff"***
 # indicamos el tamaño de las particiones, con la columna distancia
 # calcular en 10 paquetes (10 particiones). Usando función "resumen"

```
p1 <- lapply( chunk(dist,length = 10) , function(i) resumen(dist[i]))
```

Y agregar las instrucciones de ejecución paralela. Veamos

En el script se presentan diversas formas de hacer las particiones. Son experimentos para lograr la solución adecuada

También puedo hacer el corte de las particiones selectivas: ejemplo desde 1 a 100

 # calcular los 100 primeros valores en paquetes de tamaño 10

*p2 <- lapply(chunk(dist,**from=1,to=100**,by=10), function(i) resumen(dist[i]))*
crbind(p2) # para generar salidas ordenadas

*# Pero los paquetes se cortan de acuerdo al criterio de R, con "**chunk**"*

Observación: crbind() genera un buen formato de salida.

Otra forma de tratar los datos por paquetes

Puedo indicar el tamaño de los paquetes

 # calcular todo en paquetes de tamaño 100 mil ($1e^5$)
 # (Pero R no respeta el tamaño solicitado)

```
p3 <- lapply( chunk(dist,from=1,to=n,by=1e5), function(i)
resumen(dist[i]) )
crbind(p3)
```

 # tampoco los corta como yo quiero

```
# SOLUCION: le indico exactamente donde quiero que pique la partición

# calcular 500 mil exactos en paquetes de tamaño 100 mil (1e5)

p4 <- lapply( chunk(dist,from=1,to=5e5,by=1e5), function(i) resumen(dist[i]) )
crbind(p4)
# se calcula la cola restante, de 500.001 hasta n
p5 <- lapply( chunk(dist,from=5e5+1,to=n,by=1e5), function(i) resumen(dist[i]) )
crbind(p5)
```

La Ejecución Paralela

Ejecución paralela con el paquete ff, cargar la librería:

```
sfLibrary(ff)
# inicialización de los nodos con "ff"
# sfInit # para crear el background, el ambiente, cluster

        sfInit(parallel=TRUE, cpus=2, type="SOCK")

# para exportar un objeto a los esclavos (nodos). Sólo una vez en todo el proceso
# sfExport

        sfExport("dist",'resumen')
```

 # sfClusterEval(close(dist)) abre los archivos en memoria para que gc() pueda limpiar, evita problemas

sfClusterEval

sfClusterEval(open(dist)) # explicitely opening avoids a gc problema

Caso paralelo

sfLapply para ejecutar en paralelo. Una sola vez en todo el proceso, para esa variable

```
sfClusterEval(close(dist))
r2 = crbind(s2)
sfStop()
t2
```

Tamaño del archivo 2008.csv

```
> # Tamaño del archivo csv de interes en Mb
> file.size("2008.csv")/1024/1024
[1] 223.2096
>
> # MOSTRANDO LA OCUPACIÓN DE MEMORIA
>
```

```
> # Lectura comun
>
> start<-Sys.time()
> aero1 <- read_csv("2008.csv")
Rows: 2389217 Columns: 29
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr  (5): UniqueCarrier, TailNum, Origin, Dest, CancellationCode
dbl (24): Year, Month, DayofMonth, DayOfWeek, DepTime, CRSDepTime, ArrTime, CRSArrTime, FlightNum, ActualElapsedTime, C...

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
> dim(aero1)             # dimension del arreglo filas x columnas
[1] 2389217      29
> format(object.size(aero1),"Mb")   # megas que ocupa de memoria el arreglo
[1] "528.9 Mb"
>
> object.size(aero1)
554633208 bytes
> print(Sys.time() - start)
Time difference of 3.376928 secs
>
```

Tarda un poco más porque lee y crea el arreglo que contiene únicamente las direcciones de memoria, **metadata**.

```
> aero2=read.csv.ffdf(file="2008.csv",VERBOSE = TRUE,first.rows = -1,header=T)
read.table.ffdf 1..2389217 (2389217)  csv-read=10.27sec ffdf-write=1.548sec
 csv-read=10.27sec  ffdf-write=1.548sec  TOTAL=11.818sec
> dim(aero2)              # dimension del arreglo
[1] 2389217      29
> format(object.size(aero2),"Mb")   # megas que ocupa de memoria el arreglo
[1] "0.4 Mb"
>
> object.size(aero2)
463000 bytes
> print(Sys.time() - start)
Time difference of 11.81994 secs
>
```

Script aerolinea.r

El archivo original que se baja del sitio de internet contiene múltiples archivos, registros anuales. Pesa unos 1.5 Gb, comprimido.

- *Cada año son 7 millones de líneas de datos*
- *Este archivo aerolinea.csv contiene tres años de datos seleccionados del archivo original: 2005 a 2007. Tiene 21 millones de líneas.*
- *Al quitar alguna columnas quedó en 358 MB*

```
         CarrierDelay WeatherDelay NASDelay SecurityDelay LateAircraftDelay
1              16           0        0             0                0
2              NA           NA       NA            NA               NA
3              NA           NA       NA            NA               NA
4              NA           NA       NA            NA               NA
5              16           0        0             0                0
6              NA           NA       NA            NA               NA
7              NA           NA       NA            NA               NA
8              NA           NA       NA            NA               NA
:                 :            :        :             :                :
2389210        NA           NA       NA            NA               NA
2389211        NA           NA       NA            NA               NA
2389212        NA           NA       NA            NA               NA
2389213        NA           NA       NA            NA               NA
2389214        NA           NA       NA            NA               NA
2389215        NA           NA       NA            NA               NA
2389216        NA           NA       NA            NA               NA
2389217        NA           NA       NA            NA               NA
> View(aero2)
> names(aero2)
 [1] "Year"              "Month"             "DayofMonth"        "DayOfWeek"         "DepTime"
 [6] "CRSDepTime"        "ArrTime"           "CRSArrTime"        "UniqueCarrier"     "FlightNum"
[11] "TailNum"           "ActualElapsedTime" "CRSElapsedTime"    "AirTime"           "ArrDelay"
[16] "DepDelay"          "Origin"            "Dest"              "Distance"          "TaxiIn"
[21] "TaxiOut"           "Cancelled"         "CancellationCode"  "Diverted"          "CarrierDelay"
[26] "WeatherDelay"      "NASDelay"          "SecurityDelay"     "LateAircraftDelay"
> basename(filename(aero2$Year))
[1] "ffdf2b25396ab151.ff"
> basename(filename(aero2$Origin))
[1] "ffdf2b2527f67af3.ff"
> rm()
```

Y los tiempos

```
> ######## Comparacion
> ts = rbind(t1,t2,t4)[,1:3]
> dimnames(ts)[[1]] = c('secuencial','2 nodos','4 nodos')
> ts
           user.self sys.self elapsed
secuencial     0.003        0   0.003
2 nodos        0.000        0   0.013
4 nodos        0.000        0   0.012
>
>
>
>
```

Para datos tan pequeños el tiempo paralelo es >> que el tiempo secuencial

Script aerolinea.r

- *El archivo original que se baja del sitio de internet contiene múltiples archivos, registros anuales. Pesa unos 1.5 Gb, comprimido.*
- *Cada año son 7 millones de líneas de datos.*
- *Este archivo aerolinea.csv contiene tres años de datos seleccionados del archivo original: 2005 a 2007. Tiene 21 millones de líneas.*
- *Al quitar alguna columnas quedó en 358 MB*

Preparando los datos

*Para unir los tres años de datos se uso un software especial para manipular archivos de texto extensos que se llama **EmEditor** que puede acceder a más memoria que el block de notas.*

- *Con el block de notas fue imposible hacerlo, la memoria no lo permite.*
- *Tampoco con Excel, que soporta hasta 65 mil líneas por hoja.*

Vamos al script : aerolinea.r

Resultados

```
>
> ######## Comparacion
> ts = rbind(t1,t2,t4)[,1:3]
> dimnames(ts)[[1]] = c('secuencial','2 nodos','4 nodos')
> ts
           user.self sys.self elapsed
secuencial     0.089    0.011   0.099
2 nodos        0.001    0.000   0.066
4 nodos        0.001    0.000   0.043
>
>
>
>
>
>
```

Hemos incluso reducido el tiempo de ejecución

Recuerde

- Tratamos una base de datos de **tamaño inmanejable**
- Además paralelizamos el proceso
- Mejoramos los tiempos de ejecución

Tomando en cuenta la complejidad del problema hemos tenido una ganancia importante.

No podemos analizar los tiempos de la forma como veníamos haciéndolo en la primera parte de este curso

Conclusión

- *Somos capaces de tratar Big Data, algo que sin el recurso "ff" sería imposible*
- *Hemos resuelto un problema completo: manejar Big Data y ejecutar en paralelo*
- *Los resultados en tiempo de ejecución son satisfactorios*
- El manejo de la memoria es "transparente al usuario" ¡una gran ganancia!
- La lentitud del DD no ha sido un impedimento para acelerar procesos