# Informe

*Prato-Carrero*

*9 de abril de 2016*

**Planteamiento del Problema:**

Ciertas operaciones básicas sobre matrices, como el producto Matriz-vector o Matriz-Matriz, son esenciales en el cálculo de indicadores muy importantes en el análisis de grafos. Con esta asignación se pretende introducir al estudiante en elementos prácticos asociados con el manejo de matrices gigantes mediante el uso del paradigma MapReduce. El cálculo de operaciones Matriz-Vector permite la obtención de vectores como el PageRank que permite jerarquizar la importancia de las páginas web y que es la base del análisis de enlaces, la implementación de algunas redes neuronales, et al. Operaciones similares permiten hacer cálculos de centralidad importantes en la detección de, por ejemplo, influentes en las redes sociales.

**Objetivos:**

El objetivo es, dados dos matrices A , B y un vector x, realizar una función que implemente, con MapReduce, el producto Matriz-Vector cuando el vector x por su gran tamaño, no cabe en la memoria RAM.

**Acciones propuestas para la Solución:**

Para poder llevar a cabo la solución de la problemática planteada se ha elaborado un algoritmo con el paradigma MapReduce, en la que el Map se encuentra representado por las búsquedas de vectores correspondientes tanto a las filas de la(s) matrices como del vector, mientras que el reduce se encarga del cálculo de la multiplicación y suma de estos vectores previamente obtenidos para ir almacenando dichos resultados en disco, a través de archivos. Tomando en cuenta la limitante de que siempre se verifica que no se exceda el valor de la memoria limite disponible.

Las consideraciones presentes en base a la solución planteada fueron:

**-** Debido a la cantidad de variables mínimas necesarias para poder llevar acabo por lo menos una operación el tamaño de memoria limite debe ser (mayor o igual) a 336bytes. Esto sin considerar el peso de la conexiones a los archivos de datos de entrada ya que poseen un gran tamaño (lo ideal es sumar estos pesos, así el mínimo de memoria limite sería un valor mayor).

**-** Por cada asignación de variable se suma su peso object.size(), al tamaño de la memoria usada, ya que cada uno tiene un peso al momento de la ejecución, de igual manera al no

estar en uso dichas variables se restan su peso object.size(), para mantener espacio disponible en memoria.

**-** Los archivos utilizados para el apoyo en el cálculo de las operaciones, se asumen que están en disco.

**Implementación de la Solución:**

**test.r:** Archivo donde se plantea un ejemplo, tanto para la ejecución de los algoritmos de multiplicación matriz-matriz como matriz-vector. Su código Fuente está dado por:

```
#-------------------------------------------------------------------------------#
library(ProjectTemplate) # Se carga la librerias necesarias
## Comando para establecer el directorio de trabajo
setwd("C:/Users/Prato/Desktop/ProyectoICD/proyectoicd/src")
## Cargo la función de multiplicación de Matriz x Vector
source("productmv.R")
## Cargo la función de multiplicación de Matriz x Matriz
source("productmm.R")
# matriz <- "C:/Users/Manuel/Desktop/proyectoicd/data/tblAkv3x3.csv"  # Matriz de 3x3
# matriz <- "C:/Users/Manuel/Desktop/proyectoicd/data/tblAkv10x10.csv"  # Matriz de 10x10
# matriz <- "C:/Users/Manuel/Desktop/proyectoicd/data/tblAkv10x10ident.csv"  # Matriz Identidad de 10x10
# vector <- "C:/Users/Manuel/Desktop/proyectoicd/data/tblxkv3.csv"  # Vector de 3
# vector <- "C:/Users/Manuel/Desktop/proyectoicd/data/tblxkv10.csv"  # Vector de 10
matriz <- "C:/Users/Manuel/Desktop/proyectoicd/data/tblAkv10x10.csv"  # Matriz de 10x10
vector <- "C:/Users/Manuel/Desktop/proyectoicd/data/tblxkv10.csv"  # Vector de 10
N <- 10
limite_de_memoria <- 480
#Se realiza la operación de la manera básica para evidenciar que los resultados son iguales
m <- matrix(1:100,ncol=10, byrow=TRUE)
v <- 1:10
w <-  m %% v
w
productmv(matriz,vector,N,limite_de_memoria)  # Función que realiza la multiplicación de matriz-vector
matrizA <- "C:/Users/Manuel/Desktop/proyectoicd/data/tblAkv10x10.csv"
matrizB <- "C:/Users/Manuel/Desktop/proyectoicd/data/tblAkv10x10.csv"
N2 <- 10
limite_de_memoria <- 480
#Se realiza la operación de la manera básica para evidenciar que los resultados son iguales
```

```
m <- matrix(1:100,ncol=10, byrow=TRUE)

w <-  m %% m

w

productmm(matrizA,matrizB,N2,limite_de_memoria)  # Función que realiza la multiplicación de matriz-matriz

#-------------------------------------------------------------------------------#
```

**memlimit.r:** Script donde se manipula y retorna el tamaño de la memoria limite (En principio no fue necesario su uso), si es igual a -1 detecta y retorne la memoria límite de la máquina, para cualquier otro valor simplemente lo retorne. Su código Fuente está dado por:

```
#-------------------------------------------------------------------------------#

memlimit<-function(size){

  if(size==-1){

    limite<-  memory.limit()

    return (limite)

  }else{

    return (size)

  }

}

#-------------------------------------------------------------------------------#
```

**productmm.r:** Script encargado de realizar la multiplicación matriz-matriz, haciendo uso de la función productmv.r

El enfoque de esta viene dado en que para la matriz B se van tomando las columnas correspondientes a la misma, pasando a formar un vector que será usado por la función productmv.r quien recibe como parámetros los valores:

**productmv(matrizA,"C:/Users/Prato/Desktop/ProyectoICD/proyectoicd/data/vector_intermedio.csv",N,limite_de_memoria).**

Acá vector_intermedio es el mencionado, esto se realizara N veces, para las N columnas de la matriz B.

El resultado de cada operación luego de la llamada a la función productmv.r, es almacenado en un archivo de salida que se encuentra en disco denominado multiplicacionmv.csv, este último será manipulado de tal forma para almacenar en disco un nuevo archivo resultadomm.csv que contiene el resultado de la multiplicación de ambas matrices.

Es importante destacar que se toma en cuenta la simulación de la memoria para que no exceda el límite, haciendo uso además de las estructuras necesarias

Para su mayor entendimiento su código Fuente está dado por:

```
#-------------------------------------------------------------------------------#
productmm <- function(matrizA,matrizB,N,limite_de_memoria){
  memoria_usada2 <- 0
  memoria_usada2 <- memoria_usada2 + object.size(memoria_usada2)
  ## Cargo la función de multiplicación de Matriz x Vector
  source("productmv.R")
  con3 = file(matrizB, "r") # Se establece la conexión con el archivo de la matrizB
  j <- 1
  memoria_usada2<- memoria_usada2+ object.size(j)
  for(i in 1:N){
    while(j <= N){
      line2 = readLines(con3, n = 1)  # Se toma la fila del vector
      memoria_usada2<- memoria_usada2 + object.size(line2)
      line2 <- strsplit(line2, split = ",")
      i_v <- as.integer(line2[[1]][[1]])  # Posición i del vector a generar
      v_v <- as.numeric(line2[[1]][[3]])  # Valor del vector
      memoria_usada2 <-memoria_usada2 + object.size(v_v)
      memoria_usada2 <-memoria_usada2 + object.size(i_v)
      memoria_usada2 <-memoria_usada2 - object.size(line2)
      setwd("C:/Users/Prato/Desktop/ProyectoICD/proyectoicd/data")  # Se establece el directorio para el
archivo de salida
      cat(i_v,file="vector_intermedio.csv",sep=",", append = TRUE)  # Se escribe en el archivo de salida la
multiplicación
      cat(",",file="vector_intermedio.csv", append = TRUE)
      cat(v_v,file="vector_intermedio.csv",sep="\n", append = TRUE)
      memoria_usada2 <-memoria_usada2 - object.size(v_v)
      memoria_usada2 <-memoria_usada2 - object.size(i_v)
      j<-j+1;
    }
    j<-1

productmv(matrizA,"C:/Users/Prato/Desktop/ProyectoICD/proyectoicd/data/vector_intermedio.csv",N,limite
_de_memoria)  # Función que realiza la multiplicación de matriz-vector
    setwd("C:/Users/Prato/Desktop/ProyectoICD/proyectoicd/data")
    file.remove("vector_intermedio.csv")
  }
```

```
  setwd("C:/Users/Prato/Desktop/ProyectoICD/proyectoicd/reports")

  con3 = file("multiplicacionmv.csv", "r")

  line2 = readLines(con3, n = N*N)

  memoria_usada2 <-memoria_usada2 + object.size(line2)

  write.table(t(matrix(line2,ncol=N, byrow=TRUE)),file = "resultadomm.csv",col.names = F,row.names = F,
quote = F,sep = ",")

  memoria_usada2 <-memoria_usada2 - object.size(line2)

  close(con3)  # Se cierra la conexión al archivo del vector por buena práctica

  memoria_usada2 <- 0  # Se vacia la memoria usada

}

#------------------------------------------------------------------------------#
```

**productmv.r:** Script encargado de realizar la multiplicación matriz-vector.

El enfoque aplicado fue el de usar el comando *readlines()* para la lectura por filas de la matriz y del vector, con la intención de hacer el algoritmo lo más genérico posible. Este enfoque permite tener la noción de que un número N de clústers podrían estar asignados a un número N de filas a leer, de manera análoga al enfoque de dividir el límite de memoria para asignar un tamaño de memoria a la lectura por bloques. Se eligió por tanto la lectura por líneas en vez de bloques, lo que permitió optimizar el uso del límite de memoria y reducir la dificultad en la implementación de una estrategia que permitiera mantener índices para la lectura de la matriz y del vector, por lo que se redujo en gran medida el número de variables a usar, siempre con la intención de evitar ocupar más espacio en memoria.

Teniendo lo anterior en cuenta y asumiendo que el límite de memoria permitía leer al menos la primera línea, se estableció entonces un índice de filas de la matriz (*cont_filas*) para saber cuál fue la última línea leída, así por cada línea de la matriz leída, se leída un elemento del vector y luego se recorría con un ciclo buscando la siguiente línea de esa misma fila que correspondiera a la siguiente columna de la matriz, al encontrarse dicha columna, se repetía el procedimiento hasta finalizar con esa fila. Se hizo uso del comando *close(con)* siendo "*con*" la conexión usada para la lectura de la matriz, con la finalidad de poder volver a leer el archivo de la matriz desde el inicio y haciendo uso del *cont_filas* buscar hasta encontrar la siguiente fila que correspondía leer. Este procedimiento se repetía N veces, como filas tuviera la matriz.

Para la parte de la multiplicación y suma de elementos, por cada lectura de un elemento de la matriz, dicho elemento se multiplicaba con el elemento del vector recién leído y haciendo uso de un *acumulador_mult_suma* se iban acumulando el resultado para sumarlo con el siguiente resultado entre la multiplicación del próximo elemento de la matriz con el próximo elemento del vector. Al finalizar cada fila, se escribía en el archivo de salida ***multiplicacionmv.csv*** el *acumulador_mult_suma* como el primer resultado de la multiplicación y se reiniciaba el *acumulador_mult_suma* en 0, para poder usarlo con los elementos de la siguiente fila de la matriz.

Para su mayor entendimiento su código Fuente está dado por:

```
productmv <- function(matriz,vector,N,limite_de_memoria){

  #  Se asignan las cantidades de memoria RAM disponibles a los elementos que se usarán, asumiendo que cada

  #  elemento pesa como mínimo 48 bytes y una fila 96



# Consideraciones de memoria

#   size_acumulador_mult_suma <- 48  # Para mayor eficiencia, se asigna mayor memoria al vector de los elementos de la matriz que se vayan leyendo

#   size_line_matriz <- 96 # Límite de memoria asignado a la lectura de una fila del archivo de la matriz

#   size_line_vector <- 96 # Límite de memoria asignado a la lectura de una fila del archivo del vector

#   size_cont_filas <-48  # Límite de memoria asignado al cont_filas

  setwd("C:/Users/Manuel/Desktop/proyectoicd/reports")  # Se establece el directorio para el archivo de salida

   memoria_usada <- 0

   memoria_usada <- memoria_usada + object.size(memoria_usada)

  #-------------------------------------------------------------------------

  #options(warn= -1)  # Desactivar Warning, para volverlos a activar options(warn= 0)

#  file.remove("resultado_multiplicaciónmv.csv")

  acumulador_mult_suma <- 0

  memoria_usada <- memoria_usada + object.size(acumulador_mult_suma)

   cont_filas <- as.integer(1)  # Contador que permitirá saber en qué fila de la matriz estás

  memoria_usada <- memoria_usada + object.size(cont_filas)

   for(k in 1:N){  # Ciclo para recorrer hasta el final del archivo



    con = file(matriz, "r")  # Se establece la conexión con el archivo de la matriz

    #-------------------------------------------------------------------------

    # Se asume que cabe al menos el primer elemento i,j de la matriz, dado que cada elemento pesa 48 bytes

    if(cont_filas > 1){  # Caso 2: Búsqueda de la fila i-ésima

 #     cat("Estoy buscando la fila: ",cont_filas,"de la matriz\n")

      z <- 0

      memoria_usada <- memoria_usada + object.size(z)

      i_m <- ""

      while((i_m != cont_filas) && (z < N)){ # for(z in 1:n){  # Recorrer hasta encontrar la siguiente fila que corresponda leer. Nota: ésto se evitaría si se tuviera un apuntador al fichero, para comenzar desde la fila que correspondiese y no desde la primera siempre.



        #---------------Lectura del primer elemento de la matriz--------------------

        # Se asume que se tiene memoria para leer la primera fila
```

```
      line = readLines(con, n = 1)  # Se toma la fila de la matriz
      memoria_usada <- memoria_usada + object.size(line)   # 96 bytes...
      line <- strsplit(line, split = ",")
      i_m <- as.integer(line[[1]][[1]])  # Posición i de la matriz
      memoria_usada <- memoria_usada + object.size(i_m)
      #      j_m <- x[[1]][[2]]  # Posición j de la matriz
      v_m <- as.numeric(line[[1]][[3]])  # Valor de la matriz
      memoria_usada <- memoria_usada + object.size(v_m)
#       cat(i_m," ",v_m,"\n")
      memoria_usada <- memoria_usada - object.size(line)
      #--------------------------------------------------------------------------
      if(i_m == cont_filas){  # Encontré la fila que corresponde leer, es decir la siguiente columna de la fila que se está leyendo en la matriz
#         cat("Son iguales im y cont filas!\n")
        if(memoria_usada < limite_de_memoria){  # Si se cumple, quiere decir que se pueden añadir elementos al vector matriz
          #----------------Lectura del primer elemento del vector----------------------
          con2 = file(vector, "r") # Se establece la conexión con el archivo del vector
          line2 = readLines(con2, n = 1)  # Se toma la fila del vector
          memoria_usada <- memoria_usada + object.size(line2)
          line2 <- strsplit(line2, split = ",")
          #      i_v <- as.integer(y[[1]][[1]])  # Posición i del vector
          v_v <- as.numeric(line2[[1]][[2]])  # Valor del vector
          memoria_usada <-memoria_usada + object.size(v_v)  # 96 + 48 = 144 bytes
          memoria_usada <-memoria_usada - object.size(line2)
#            cat(v_v,"\n")
          #       Sys.sleep(2)
          #----------------Multiplicación y Suma de los elementos leídos----------------
          acumulador_mult_suma <- acumulador_mult_suma + (v_m * v_v)
          memoria_usada <- memoria_usada + object.size(acumulador_mult_suma)  # 144 + 48 = 192 bytes
          memoria_usada <- memoria_usada - (object.size(v_m) + object.size(v_v))
 #          cat("Multiplico el ",v_m,"con el ",v_v,"\n")
        }
      }
      z <- z + 1
    }
    memoria_usada <- memoria_usada - object.size(z)
```

```
  }else{  # Caso 1: 1ra Fila
    #----------------Lectura del primer elemento de la matriz----------------------
    # Se asume que se tiene memoria para leer la primera fila
    line = readLines(con, n = 1)  # Se toma la fila de la matriz
    memoria_usada <- memoria_usada + object.size(line)   # 96 bytes...
#     cat("Caso 1ra Fila",line,"\n")
    line <- strsplit(line, split = ",")
    i_m <- as.integer(line[[1]][[1]])  # Posición i de la matriz
    memoria_usada <- memoria_usada + object.size(i_m)
#     j_m <- x[[1]][[2]]  # Posición j de la matriz
    v_m <- as.numeric(line[[1]][[3]])  # Valor de la matriz
    memoria_usada <- memoria_usada + object.size(v_m)
#     cat(i_m," ",v_m,"\n")
    memoria_usada <- memoria_usada - object.size(line)
    #----------------Lectura del primer elemento del vector----------------------
    con2 = file(vector, "r") # Se establece la conexión con el archivo del vector
    line2 = readLines(con2, n = 1)  # Se toma la fila del vector
    memoria_usada <- memoria_usada + object.size(line2)

    line2 <- strsplit(line2, split = ",")
#     i_v <- as.integer(y[[1]][[1]])  # Posición i del vector
    v_v <- as.numeric(line2[[1]][[2]])  # Valor del vector
    memoria_usada <-memoria_usada + object.size(v_v)  # 96 + 48 = 144 bytes
    memoria_usada <-memoria_usada - object.size(line2)
#     cat(v_v,"\n")
    #       Sys.sleep(2)
# Nota: La lectura del elemento de la matriz y del elemento del vector, se podría paralelizar para optimizar
# el algoritmo, mediante el paquete parallel.r: https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf
# library(parallel)

    #----------------Multiplicación y Suma de los elementos leídos----------------
    acumulador_mult_suma <- acumulador_mult_suma + (v_m * v_v)
    memoria_usada <- memoria_usada + object.size(acumulador_mult_suma)  # 144 + 48 = 192 bytes
    memoria_usada <- memoria_usada - (object.size(v_m) + object.size(v_v))
 #    cat("Multiplico el ",v_m,"con el ",v_v,"\n")
  }
```

```
    i <- 0
    memoria_usada <- memoria_usada + object.size(i)
    while(i < ((N*N) - cont_filas)){  # Ciclo para recorrer el resto de las filas (columnas restantes de la fila que se está leyendo). i in 2 = para que comience en 1, ya que i in 1 inicia en 0, ésto para que lea la siguiente fila a la que leyó anteriormente (1ra fila o fila i-ésima caso 2). Se decrementará a medida que el cont_filas aumente.
      #----------------Lectura del primer elemento de la matriz--------------------
      # Se asume que se tiene memoria para leer la primera fila
      line = readLines(con, n = 1)  # Se toma la fila de la matriz
      memoria_usada <- memoria_usada + object.size(line)   # 96 bytes...
#      cat(line,"\n")
      line <- strsplit(line, split = ",")
      i_m <- as.integer(line[[1]][[1]])  # Posición i de la matriz
      memoria_usada <- memoria_usada + object.size(i_m)

      #      j_m <- x[[1]][[2]]  # Posición j de la matriz
      v_m <- as.numeric(line[[1]][[3]])  # Valor de la matriz
      memoria_usada <- memoria_usada + object.size(v_m)
#      cat(i_m," ",v_m,"\n")
      memoria_usada <- memoria_usada - object.size(line)
      #------------------------------------------------------------------------
      if(cont_filas == i_m) {
       if(memoria_usada < limite_de_memoria){  # Si se cumple, quiere decir que se pueden añadir elementos al vector matriz
         #----------------Lectura del primer elemento del vector----------------------
         line2 = readLines(con2, n = 1)  # Se toma la fila del vector
         memoria_usada <- memoria_usada + object.size(line2)
         line2 <- strsplit(line2, split = ",")
         #      i_v <- as.integer(y[[1]][[1]])  # Posición i del vector
         v_v <- as.numeric(line2[[1]][[2]])  # Valor del vector
         memoria_usada <-memoria_usada + object.size(v_v)  # 96 + 48 = 144 bytes
         memoria_usada <-memoria_usada - object.size(line2)
#          cat(v_v,"\n")
         #       Sys.sleep(2)
         #----------------Multiplicación y Suma de los elementos leídos----------------
         acumulador_mult_suma <- acumulador_mult_suma + (v_m * v_v)
         memoria_usada <- memoria_usada + object.size(acumulador_mult_suma)  # 144 + 48 = 192 bytes
         memoria_usada <- memoria_usada - (object.size(v_m) + object.size(v_v))
```

```
#          cat("Multiplico el ",v_m,"con el ",v_v,"\n")
      }
     }
     i <- i + 1
   }
   memoria_usada <- memoria_usada - object.size(i)

   cont_filas <- cont_filas + 1  # Incremento para recorrer ahora la siguiente fila de la matriz
#    cat("cont_filas",cont_filas,"\n")
   cat(acumulador_mult_suma,file="multiplicacionmv.csv",sep="\n", append = TRUE)  # Se escribe en el archivo de salida la multiplicación
   acumulador_mult_suma <- 0
   close(con)  # Se cierra la conexión al archivo de la matriz para empezar otra vez desde el inicio
   close(con2)  # Se cierra la conexión al archivo del vector para empezar otra vez desde el inicio
 }
 #close(con)  # Se cierra la conexión al archivo de la matriz por buena práctica
 #close(con2)  # Se cierra la conexión al archivo del vector por buena práctica
 memoria_usada <- 0  # Se vacía la memoria usada
}
```

**Conclusiones:**

Hay dos puntos a destacar que fueron importantes a la hora de implementar las estrategias abordadas para las soluciones de los algoritmos, el primero es la manera que se tomó para la lectura de los archivos, siendo ésta por líneas, ya que la elección de ésta permitió reducir la complejidad de los algoritmos; el segundo es la constante presencia de hacer eficiente el algoritmo, más de lo que se suele hacer a la hora de desarrollar cualquier algoritmo, puesto a que se tenía un argumento *límite de memoria* presente. Ambos puntos quedarán presentes a la hora de desarrollar soluciones para problemas de manejo de grandes volúmenes de datos, siendo considerados como buena experiencia de éste proyecto.

Otro tema a resaltar fue el hecho de poder implementar soluciones usando el enfoque Map Reduce, bajo R sin usar la librería *library(rmr2)* aplicada en el entorno Hadoop, lo que permite obtener aún más experiencia en el ámbito profesional, puesto a que se pueden ofrecer soluciones eficientes de implementaciones usando éste enfoque.