Reporte práctica 6

Sistema multiagente

A 19 de Septiembre de 2017

Sistema multiagente

Un sistema multiagente es todo aquel sistema compuesto por más de un agente inteligente y que además pueden interactuar los unos con los otros.[1] Este tipo de sistema se utiliza para resolver problemas en las áreas del comercio online, [2] la respuesta a desastres (naturales)[3] y el modelado de estructuras sociales.[4]

Práctica

En esta práctica se analizó el código de un sistema multiagente, el cual tiene como función analizar la forma en que una epidemia se esparce, mejor dicho, la forma en que una enfermedad se puede esparcir en una población.

Cómo práctica se analizó el código para conocer las líneas de comando que se pueden modificar aplicando paralelización.

El código original se modificó de la siguiente manera, como se muestra en la figura 1.

```
1. l <- 1.5
2. n <- 50
3. pi <- 0.05
4. pr <- 0.02
5. v <- l / 30
6. r <- 0.1
7. tmax <- 100
8.
9. suppressMessages(library(doParallel))
10.       suppressMessages(library(foreach))
11.       nu<- makeCluster(2)
12.       registerDoParallel(nu)
```

Fig. 1. Fragmento inicial modificado del código original.

Como se ve en la figura 1 todas las variables se trasladaron al inicio del código, y se agregaron las instrucciones “suppressMessages(library)” para los paquetes “foreach” y “doParallel”,[5] los cuales se utilizaron para la paralelización del código, además de esto se creó un cluster llamado “nu” el cual utiliza dos núcleos para llevar a cabo la paralelización.

En la figura 2 se muestra la siguiente parte que se modificó el código.

```
1.  Actulizaciones <- function(i) { # movimientos y actualizaciones
2.    a <- agentes[i, ]
3.    if (contagios[i]) {
4.      a$estado <- "I"
5.    } else if (a$estado == "I") { # ya estaba infectado
6.      if (runif(1) < pr) {
7.        a$estado <- "R" # recupera
8.      }
9.    }
10.         a$x <- a$x + a$dx
11.         a$y <- a$y + a$dy
12.         if (a$x > l) {
13.            a$x <- a$x - l
14.         }
15.         if (a$y > l) {
16.            a$y <- a$y - l
17.         }
18.         if (a$x < 0) {
19.           a$x <- a$x + l
20.         }
21.         if (a$y < 0) {
22.           a$y <- a$y + l
23.         }
24.         return(a)
25.       }
26.       agentes <- foreach(i = 1:n, .combine = rbind) %dopar%
   Actulizaciones(i)
27.       aS <- agentes[agentes$estado == "S",]
28.       aI <- agentes[agentes$estado == "I",]
29.       aR <- agentes[agentes$estado == "R",]
```

Fig. 2. Fragmento de actualizaciones modificado del código original.

En la figura dos observamos la creación de una función en “i” nombrada “Actulizaciones”, la cual nos ayuda para agregar el comando “foreach”, éste nos ayuda a hacer las repeticiones, en la primer parte del comando se colocó el intervalo en el que se trabajó, se agregó una función “.combine” para que está nos arrojará los datos en forma de matriz al usar el comando “rbind”, al final del comando foreach se agregaron las funcionas “%dopar%” y “Actulizaciones(i)” para que hiciera los cálculos en paralelo y guardar los datos en la función i, esto se realizó tomando de ejemplo el código de la práctica 3 del curso de simulación y la opción de ayuda de RStudio.[5, 6]

Reto 1

En el reto 1 con nuestro código ya paralelizado se agregó una variable nombrada “pv”, la cual representa la probabilidad de agentes vacunados. Esta variable se utilizó para dar el estado de recuperado a los agentes y así no se contagiaran ni propagarán la infección.

En la figura 4 se muestra el código modificado para el reto 1.

```
l <- 1.5
n <- 50
pi <- 0.05
pr <- 0.02
v <- l / 30
r <- 0.1
tmax <- 100
pv <- 0.1
suppressMessages(library(doParallel))
      suppressMessages(library(foreach))
      nu<- makeCluster(2)
      registerDoParallel(nu)
```

Fig. 4. Código paralelizado para probabilidad de vacunados.

En la figura 4 observamos que se ha introducido la variable “pv” para agregar la probabilidad de agentes vacunados, también observamos que se mantuvieron las líneas de comando de los paquetes “doParallel” y “foreach”, así como la creación del cluster de dos núcleos.

```
agentes <- data.frame(x = double(), y = double(), dx = double(),
dy = double(), estado  = character())
for (i in 1:n) {
 e <- "S"
 if (runif(1) < pi) {
   e <- "I"
 }
 if (runif(1) < pv) {
   e <- "R"
 }
       agentes <- rbind(agentes, data.frame(x = runif(1, 0, l),
y = runif(1, 0, l),
                                            dx = runif(1, -v, v),
dy = runif(1, -v, v),
                                            estado = e))
       levels(agentes$estado) <- c("S", "I", "R")
      }
```

Fig. 5. Código con probabilidad pv.

En la figura 5 podemos observar la manera en que se agregó la probabilidad de agentes vacunados para que estos actuaran al inicio del código y evitaran la propagación de la infección. Esto se logró al agregar dentro del “for” un “runif” con valores menores a la probabilidad de vacuna y dándoles el estado de “recuperados”.

Resultados

Como resultado de la examinación se obtuvieron tiempos similares en las corridas del código sin paralelizar y el paralelizado, al tener tiempos de 13.870 y 13.750 segundos respectivamente. Esto indica que la paralelización puede beneficiar o perjudicar los tiempos de operación de los códigos.

En la figura 6 se muestra el gráfico de puntos del tiempo en el que se propaga la infección.

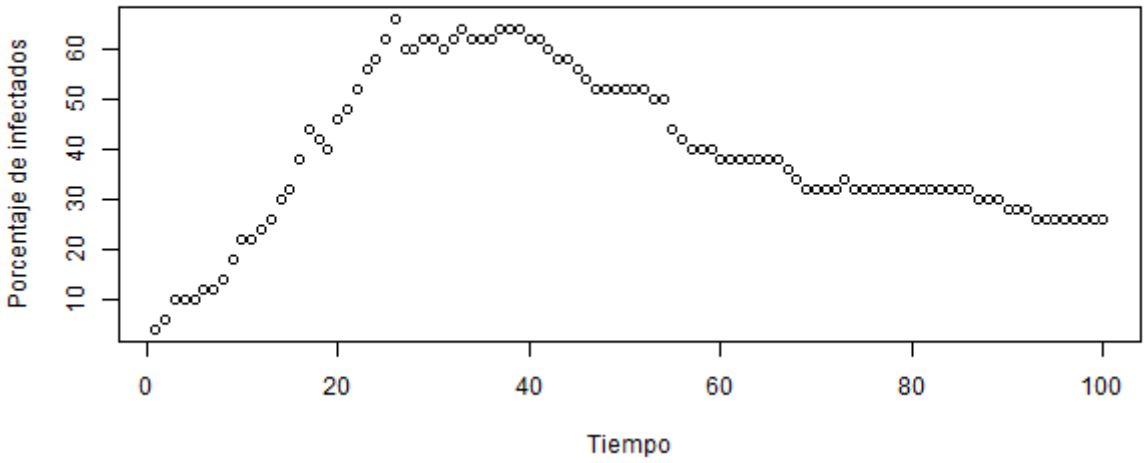

Fig. 6. Gráfico de puntos del tiempo contra el porcentaje de infectados, sin probabilidad de vacuna.

En la figura 6 se observa el grafico de la propagación de la infección con un tiempo de 100 pasos y una población de 50 agentes donde se observa como la infección aumenta de manera gradual y esta a su vez disminuye a partir del paso 40 al existir agentes que se han recuperado de la infección.

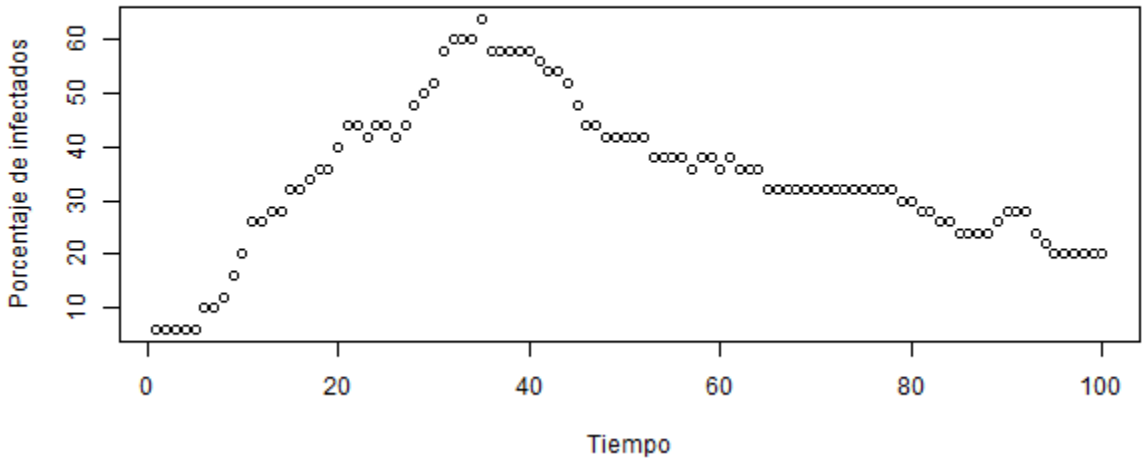

Fig. 7. Gráfica de puntos del tiempo contra el porcentaje de infectados con una probabilidad de vacuna.

En la figura 7 se observa el grafico de la propagación de la infección cuando existe una probabilidad de agentes con una vacuna, se aprecia como el tiempo para llegar al máximo de infectados aumenta y como el número de recuperados aumenta.

Conclusiones

De esta práctica se puede concluir que los sistemas multiagente son de gran importancia, ya que sin estos situaciones como la propagación de infecciones serían tareas muy difíciles de realizar. De la práctica se concluye que el paralelizar no siempre disminuye los tiempos de ejecución de los códigos, sino que al contrario esto puede ser perjudicial, por esta razón es importante analizar bien el código y ver si es factible realizar una paralelización utilizando los distintos paquetes con los que cuenta R. Del reto 1 se concluye que el comportamiento de la propagación de las infecciones puede variar de una forma drástica al agregar una probabilidad que permite tener una inmunidad mediante el uso de una vacuna.

Bibliografía

1) Mas, Ana, (2004), *Agentes software y sistemas multiagente: conceptos, arquitecturas y aplicaciones*, Ed. Pearson Educación.
2) Rogers, A., E., David, J., Schiff, N. R., Jennings, (2007), *The effects of proxy bidding and minimum bid increments within eBay Auctions*, *ACM Computing Surveys*.
3) Nathan Schurr, Janusz Marecki, (2005), *The future of disaster response: Humans working with multiagents teams using DEFACTO.*
4) Ron Sun, Isaac Naveh, *Simulating organizational decision-making using a cognitively realistic agent model*, *Journal of Artificial Societies and Social Simulation.*
5) http://elisa.dyndns-web.com/teaching/comp/par/p3.html
6) rstudio-da6580.pdf