Reporte práctica 8

Modelo de urnas

A 03 de Octubre de 2017

Modelo de urnas

Un modelo de urnas se basa en la construcción de un conjunto de urnas con bolas de diferentes colores en su interior, después se establecen las reglas para determinar el o los procedimientos a seguir según el color de la bola. Este modelo es útil para medir la aleatoriedad de distintos sistemas.1

Práctica

En esta práctica se abordan los fenómenos de coalescencia y fragmentación, donde las partículas se unen y se descomponen para formar cúmulos.

Lo primero que se realizó en esta práctica fue la paralelización del código original.

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

```
1. suppressMessages(library(testit)) # para pruebas
2. suppressMessages(library(doParallel))
3. suppressMessages(library(foreach))
4. registerDoParallel(makeCluster(2))
```

Fig. 1. Paquetes activados.

En la figura 1 se muestran los paquetes que se activaron para la paralelización del código, y se creó un *cluster* que utiliza dos núcleos para llevar a cabo la paralelización, este número de núcleos se definió en base a la capacidad de la computadora.

```
1. k <- 10000
2. n <- 1000000
```

Fig. 2. Variables de código.

En la figura 2 se muestran las variables donde *k* es el número de cúmulos y *n* la cantidad de partículas en el sistema.

```
1. separar <- function(){
2.  cumulos <- integer()
3.  urna <- freq[i,]
4.  if (urna$tam > 1) { # no tiene caso romper si no se puede
5.     cumulos <- c(cumulos, romperse(urna$tam, urna$num))
```

```
6.  } else {
7.    cumulos <- c(cumulos, rep(1, urna$num))
8.  }
9.  return(cumulos)
10.      }
11.
12.      reunir <- function(){
13.       cumulos <- integer()
14.       urna <- freq[i,]
15.       cumulos <- c(cumulos, unirse(urna$tam, urna$num))
16.       return(cumulos)
17.      }
18.
19.      syr <- function(){
20.       cumulos <- juntarse[2*i-1] + juntarse[2*i]
21.       return(cumulos)
22.      }
23.
24.      for (paso in 1:duracion) {
25.
26.       assert(sum(cumulos) == n)
27.       cumulos <- foreach(i = 1:dim(freq)[1], .combine = c) %dopar%
   separar()
28.
29.       assert(sum(cumulos) == n)
30.       assert(length(cumulos[cumulos == 0]) == 0) # que no haya vacios
31.       freq <- as.data.frame(table(cumulos)) # actualizar urnas
32.       names(freq) <- c("tam", "num")
33.       freq$tam <- as.numeric(levels(freq$tam))[freq$tam]
34.       assert(sum(freq$num * freq$tam) == n)
35.
36.       cumulos <- foreach(i=1:dim(freq)[1], .combine = c) %dopar%
   reunir ()
37.
38.       assert(sum(abs(cumulos)) == n)
39.       assert(length(cumulos[cumulos == 0]) == 0) # que no haya vacios
40.       juntarse <- -cumulos[cumulos < 0]
41.       positivos <- cumulos[cumulos > 0]#
42.        assert(sum(positivos) + sum(juntarse) == n)
43.        nt <- length(juntarse)
44.       if (nt > 0) {
45.          if (nt > 1) {
46.            juntarse <- sample(juntarse)
47.
48.           cumuloss <- foreach(i = 1:floor(nt/2), .combine = c) %dopar%
   syr()
49.           cumulos <- c(positivos, cumuloss)
50.         }
51.
```

Fig. 3. Funciones aplicadas al código.

En la figura 3 se crearon 3 funciones para paralelizar los cálculos para establecer la ruptura, la unión y la combinación de éstas tres. Para la parte de la ruptura se creó una función *separar*, para

la unión se creó la función *reunir* y para la combinación de las dos funciones se creó la función *syr*, y se agregaron un *foreach* para cada una de las funciones creadas.

```
codigo <- data.frame()
tiempo <- data.frame()

for(i in 1:5){
 source('C:/Users/Pablo/Desktop/sim/p8/p8_6.R', encoding = 'UTF-8')
 source('C:/Users/Pablo/Desktop/sim/p8/p8_6mod.R', encoding = 'UTF-8')
 tiempo <- cbind(totalo, total)
 codigo <- rbind(codigo, tiempo)
}
      colnames(codigo) <- c("C. Original", "C. Modificado")
      boxplot(codigo, col = c("orange", "green"))
```

Fig. 4. Código para medir la diferencia de tiempos del código original y el modificado.

En la figura 4 se observa el código para medir la diferencia de tiempos entre el código original y el código modificado, en se crearon dos *data.frame*, uno para guardar el tiempo de los diferentes códigos. Se utilizó la función *source* para mandar llamar los dos códigos y así correrlos 5 veces y así poder imprimir un gráfico caja-bigote de la diferencia de tiempo de los códigos.

Resultados

Como resultado se obtuvieron histogramas donde varía la frecuencia de los tamaños de los cúmulos con el número de pasos.

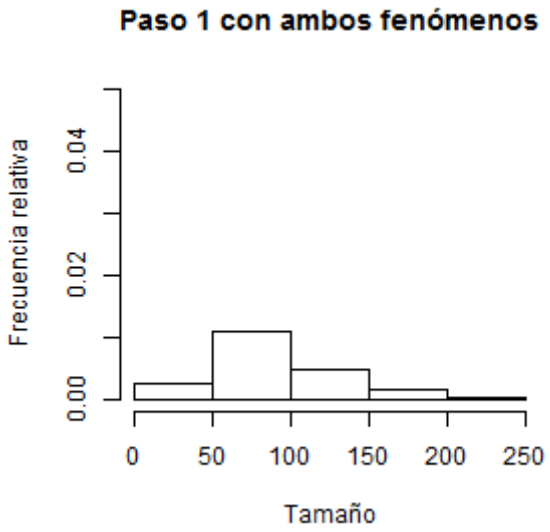

Fig. 5. Histograma de tamaño contra frecuencia relativa.

En la figura 5 tenemos el histograma del tamaño contra la frecuencia relativa donde se puede observar que la frecuencia relativa es mayor para tamaños de cúmulos menores, esto se debe a la

dificultad de las partículas para seguir uniéndose y prefieren romperse y volver a unirse para formar cúmulos de menor tamaño, esto explica porque la frecuencia relativa es mayor para tamaños menores.

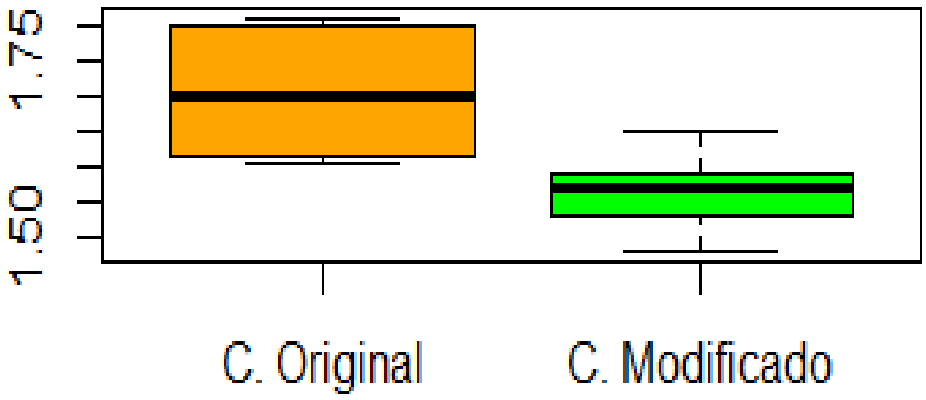

Fig. 6. Gráfico caja-bigote de diferencia de tiempos.

En la figura 6 se muestra el gráfico caja-bigote de los tiempos donde se observa que el tiempo es mayor para el código original mientras que es menor para el código modificado, este gráfico caja-bigote se realizó con un valor de *k* de 100,000 cúmulos el cuál es el número de cúmulos con el que se observa una diferencia en los tiempos de los códigos, ya que a valores bajos de *k* el código original es más rápido que el código modificado debido a la paralelización que existe en éste.

Conclusiones

En la práctica 8 se ha llegado a la conclusión que al tener paralelizar el código original se pueden obtener tiempos de ejecución menores que en el código sin paralelizar. Pero para obtener esa disminución en tiempos debe haber un tamaño de muestra, ya que al repetir el código con tamaños de muestra mayores el tiempo de ejecución disminuye.

Reto 1

En el reto 1 hay que demostrar de manera estadística si el código modificado es viable para diferentes valores de *k* y cambiando la población de partículas *n* a *30k*.

```
1. resultado <- data.frame()
2. To <- numeric()
3. Tm <- numeric()
4. 
5. for (k in seq(80000, 100000, 10000)){
6.   for (i in 1:3){
```

```
7.      source('C:/Users/Pablo/Desktop/sim/p8/reto1/p8_6sink.R', encoding =
    'UTF-8')
8.      To <- cbind(k, "o", totalo)
9.
10.           source('C:/Users/Pablo/Desktop/sim/p8/reto1/p8_6modsink.R', e
   ncoding = 'UTF-8')
11.           Tm <- cbind(k, "m", totalo)
12.           resultado <- rbind(resultado, To, Tm)
13.         }
14.       }
15.
16.       names(resultado) <- c("k","tipo","Tiempo")
17.       resultado$k <- as.numeric(levels(resultado$k))[resultado$k]
18.       resultado$Tiempo <- as.numeric(levels(resultado$Tiempo))[resultad
   o$Tiempo]
19.       resultado$tipo <- as.factor(resultado$tipo)
20.       resultado[resultado$Tiempo>10,3] <- resultado[resultado$Tiempo>10
   ,3]/60
21.       resultado$k <- resultado$k/1000
22.       png("tiempos.png", width = 600, height = 800, pointsize = 15)
23.       boxplot(Tiempo ~
   tipo * k, data = resultado, col = c("orange", "green"), xlab = "Valores
   de K (10^3)", ylab = "Tiempo (min)")
24.       legend("topleft", inset = 0.02, c("C. Original", "C.
   Modificado"), fill = c("orange", "green"), horiz=TRUE, cex=0.8, box.lty
    = 0)
25.       graphics.off()
26.
27.       for (k in seq(100, 200, 50)){
28.         PTo <- resultado[resultado$k == k & resultado$tipo=="o",]
29.         PTm <- resultado[resultado$k == k & resultado$tipo=="m",]
30.         vO <- PTo$Tiempo
31.         vm <- PTm$Tiempo
32.         examen <- t.test(vO, vm)
33.         print(examen)
34.       }
```

Fig. 7. Código para el reto 1.

En la figura 7 se muestra el código para el reto 1, en este se utilizo el código de la tarea, en donde se elimino la variable *k* de los códigos que se utilizaron con la función *source*. Con la ayuda de RStudio se creo un gráfico caja-bigotes, donde se agrego el comando legend para agregar una leyenda y diferenciar los distintos gráficos caja-bigote. Se creo un *for* donde *k* toma distintos valores. Para realizar la prueba estadística se utilizo el comando *t.test* para comparar dos vectores que creamos para poder compararlos, esto se realizó con ayuda de victor ortiz.[2, 3]

Resultados

Como resultados se obtuvo un gráfico caja-bigote y se obtuvo un resultado de la prueba t.test.

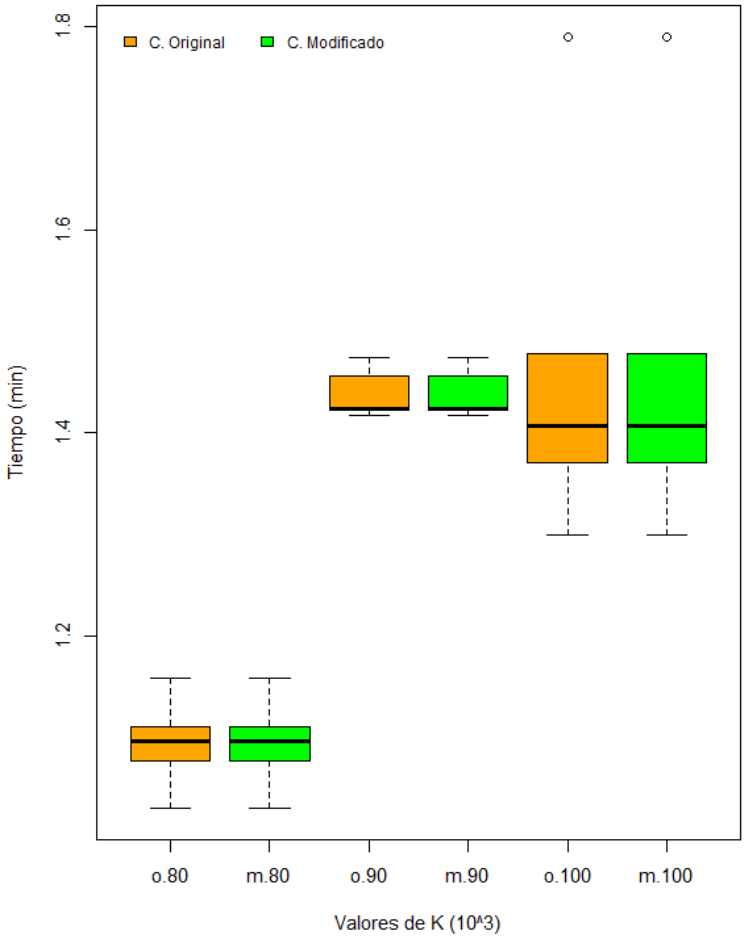

Fig. 8. Gráfico caja-bigotes con varios valores de *k*.

En este gráfico se observa que el número de cúmulos no fue suficiente para observar un cambio significativo en el tiempo de ejecución del código, por eta razón las cajas-bigote de los códigos original y modifcado son similares en sus tiempos de ejecución.

Para la prueba *t.test* se obtuvo un valor de 1.05 el cual representa que el ahorro del tiempo de ejecución no es significativo, al menos para el número de cúmulos 80,000, 90,000 y 100,000utilizados para nuestra población de *n=30k*.

Bibliografía

1) https://ocw.uca.es/pluginfile.php/294/mod_resource/content/1/Los%20modelos%20de%20urnas.pdf

2) VictorOrtiz0320, código reto1.
3) Documentación de R para uso de source.