Teoría de colas

Edson Raúl Cepeda Márquez

12 de febrero de 2019

1. Objetivo

El objetivo principal de esta practica es examinar el ordenamiento de trabajos, y analizar los tiempos de ejecución cuando se varia la cantidad de núcleos asignados al cluster. Se paraleliza la tarea de calcular numero primos y no primos para así poder analizar los tiempos de ejecución dependiendo de la proporción y la magnitud de los números primos y no primos. Se hace uso de un archivo descargado que contiene numero primos, así como también el código escrito en el lenguaje de programación R [1] y el material didáctico de apoyo, todo esto disponible en la pagina de la Dra. Elisa Schaeffer [2].

Como cualquier experimento este se repetirá un determinado numero de veces para poder comparar entre replicas y visualizar los resultados.

2. Desarrollo

Para comenzar es necesario descargar el archivo que contiene los números primos, hay tener en cuenta la potencia del computador en la que se este ejecutando este código puesto que puede llegar a ser un tanto pesado. Ejecutar las funciones del cálculo de números primos con cantidades grandes de números o magnitudes muy elevadas pueden costar más al computador, sobre todo si no se cuenta un cantidad considerable de núcleos para la paralelización de esta tarea.

Primero se leen y se extraen los números del archivo y se acomodan en un arreglo:

```
#Leer archivo de numeros primos
lista <- read.csv("C:\\Users\\OrbitalCardinal\\Desktop\\Practica 3\\Primos 10k - 20k.txt", sep="",
    header=FALSE)
```

Se procede a colocar los números dentro de un vector:

```
#Extraer los numeros primos del archivo en un vector
nums <- c()
for(i in 1:129)
{
  for(j in 1:8) {
    nums <- c(nums,lista[i,j])
  }
}
```

Cabe recalcar que estaremos calculando números primos y no primos en un rango aproximado de 10,000 números. A continuación se realiza la separación y creamos dos vectores para diferenciar cuales son números primos y cuales no lo son:

```
#Crear vectores de primos y no primos
primos <- c()
```

```
noprimos <- c()
desde <- nums[1]
hasta <-  nums[length(nums)]
for(i in desde:hasta) {
  if(primo(i) == TRUE) {
    primos <- c(primos,i)
  }
  if(primo(i) == FALSE) {
    noprimos <- c(noprimos,i)
  }
}
```

En la siguiente linea de código se adecua el vector de los números no primos para que tenga la misma longitud que el vector de números primos:

```
noprimos <- sample(noprimos[1]:noprimos[length(noprimos)], length(primos))
```

Como se quiere analizar la proporción y la magnitud de los números primos y no primos a la vez que examinamos los tiempos de ejecución al variar el numero de núcleos asignados al cluster, creamos 3 nuevos vectores en los cuales se reparten los números primos y no primos que anteriormente se habían calculado, y se asignan proporciones en base a la cantidad de números primos:

```
#Mezclamos los vectores variando las proporciones de primos y no primos dentro de el
#25/75
mvectores1 <- c()
mvectoresTEMP1 <- c()
mvectoresTEMP1 <- c(mvectoresTEMP1,sample(primos,length(primos)*0.25))
mvectoresTEMP1 <- c(mvectoresTEMP1, sample(noprimos,length(noprimos)*0.75))
mvectores1 <- sample(mvectoresTEMP1)

#50/50
mvectores2 <- c()
mvectoresTEMP2 <- c()
mvectoresTEMP2 <- c(mvectoresTEMP2,sample(primos,length(primos)*0.5))
mvectoresTEMP2 <- c(mvectoresTEMP2, sample(noprimos,length(noprimos)*0.5))
mvectores2 <- sample(mvectoresTEMP2)

#75/25
mvectores3 <- c()
mvectoresTEMP3 <- c()
mvectoresTEMP3 <- c(mvectoresTEMP3,sample(primos,length(primos)*0.75))
mvectoresTEMP3 <- c(mvectoresTEMP3, sample(noprimos,length(noprimos)*0.25))
mvectores3 <- sample(mvectoresTEMP3)
```

Las proporciones varían de manera que queda un vector con la mitad de números primos, otro con un cuarto de los números primos, y otro con tres cuartos de los números primos. También se establecen los rangos para cada una de las proporciones colocándolos en los vectores que posteriormente serán evaluados, el resto serán numeros no primos:

```
#Rangos del vector con proporcionalidad 50% primos / 50% no primos
original1 <- mvectores1[1]:mvectores1[length(mvectores1)]
invertido1 <- mvectores1[length(mvectores1)]:mvectores1[1]

#Rangos del vector con proporcionalidad 25% primos / 75% no primos
original2 <- mvectores2[1]:mvectores2[length(mvectores2)]
invertido2 <- mvectores2[length(mvectores2)]:mvectores2[1]

#Rangos del vector con proporcionalidad 75% primos / 25% no primos
original3 <- mvectores3[1]:mvectores3[length(mvectores3)]
invertido3 <- mvectores3[length(mvectores3)]:mvectores3[1]
```

Se establecen el numero de núcleos a usar en este caso son 4 núcleos pero se resta uno para dejar ese núcleo libre para el sistema operativo y evitar inconvenientes a la hora de ejecutar el código:

```
nucleos <- detectCores(logical = FALSE)-1 #Numero de nucleos del computador
```

También se establece el número de replicas del experimento:

```
replicas <- 10 #Numero de veces a repetir el experimento
```

Empiezan los ciclos y a continuación se hace uso de una serie de comparaciones para detectar la cantidad de núcleos que esta usada en el momento de la iteración, para posterior mente guardar los datos en variables con nombres que ayudan a identificar a que grupo pertenecen y que resultaron de evaluar el rango de números dependiendo de su proporción:

```
for(p in 1:3) { #Opciones de proporcion 1- 25%, 2- 50%, 3- 75%
  if(p == 1) {
    for(nuc in nucleos:1) { #Variacion el numero de nucleos
      registerDoParallel(makeCluster(detectCores() - nuc))

      for (r in 1:replicas) { #Repetir el experimento cuantas veces la variable replica indique

        if(nuc == 1) { #comparando si se asigno 1 nucleo
          Aot1 <- c(Aot1, system.time(foreach(n = original1, .combine=c) %dopar% primo(n))[3]) # de menor a mayor
          Ait1 <- c(Ait1, system.time(foreach(n = invertido1, .combine=c) %dopar% primo(n))[3]) # de mayor a menor
          Aat1 <- c(Aat1, system.time(foreach(n = sample(original1), .combine=c) %dopar% primo(n))[3]) # orden aleatorio

        }
        if(nuc == 2) {
          Aot2 <- c(Aot2, system.time(foreach(n = original1, .combine=c) %dopar% primo(n))[3]) # de menor a mayor
          Ait2 <- c(Ait2, system.time(foreach(n = invertido1, .combine=c) %dopar% primo(n))[3]) # de mayor a menor
          Aat2 <- c(Aat2, system.time(foreach(n = sample(original1), .combine=c) %dopar% primo(n))[3]) # orden aleatorio

        }
        if(nuc == 3) {
          Aot3 <- c(Aot3, system.time(foreach(n = original1, .combine=c) %dopar% primo(n))[3]) # de menor a mayor
          Ait3 <- c(Ait3, system.time(foreach(n = invertido1, .combine=c) %dopar% primo(n))[3]) # de mayor a menor
          Aat3 <- c(Aat3, system.time(foreach(n = sample(original1), .combine=c) %dopar% primo(n))[3]) # orden aleatorio

        }


      }
      stopImplicitCluster()
    }
  }

  if(p == 2) { #comprobando si se asigno 2 nucleos
    for(nuc in nucleos:1) { #Variacion el numero de nucleos
      registerDoParallel(makeCluster(detectCores() - nuc))

      for (r in 1:replicas) { #Repetir el experimento cuantas veces la variable replica indique

        if(nuc == 1) {
```

```
40          Bot1 <- c(Bot1, system.time(foreach(n = original2, .combine=c) %dopar% primo(n))[3]) # de
    menor a mayor
41          Bit1 <- c(Bit1, system.time(foreach(n = invertido2, .combine=c) %dopar% primo(n))[3]) # de
    mayor a menor
```

Se guardan cada uno de los datos resultantes en tres nuevos arreglos, los cuales contendrán cada uno de los tiempos de ejecución, para cada uno de los núcleos. Estos tres arreglos representaran los resultados de medir estas cantidades para las tres proporciones de números primos antes mencionadas.

Finalmente se acomodan los datos de estos últimos arreglos de manera que puedan ser graficados.

```
1 Dnucleos1 <- data.frame(Nucleos=c(rep(1,Ndatos),rep(2,Ndatos),rep(3,Ndatos)), Orden= c("1","2","3"),
    Tiempos= tiempos1)
2 Dnucleos2 <- data.frame(Nucleos=c(rep(1,Ndatos),rep(2,Ndatos),rep(3,Ndatos)), Orden= c("1","2","3"),
    Tiempos= tiempos2)
3 Dnucleos3 <- data.frame(Nucleos=c(rep(1,Ndatos),rep(2,Ndatos),rep(3,Ndatos)), Orden= c("1","2","3"),
    Tiempos= tiempos3)
```

3. Resultados y conclusiones

Ahora se tiene manera de visualizar los datos y examinar los resultados.

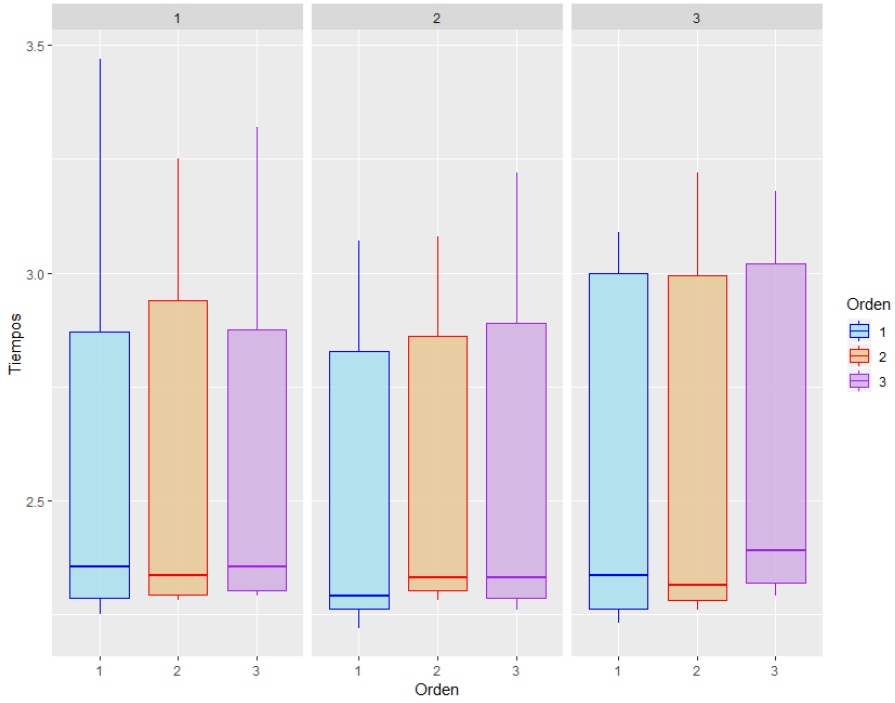

Figura 1: Tiempo total de ejecución dependiendo del orden del arreglo y la cantidad de núcleos para una proporción de 1/4.

Como se puede observar para esta proporción 1 y para los 3 núcleos, los tiempos de ejecución tienen un rango muy

parecido, estos vienen desde un poco menos de 2.5 hasta casi alcanzar 3.0 en la mayoría de las replicas, lo que quiere decir que para esta proporción 3/4 de los números que no eran primos estaban ralentizando el calculo y se interponían a la vez que generaban un retraso, pues estos ocupaban la mayor parte del arreglo y la función seguía evaluando hasta comprobar si era primo o no.

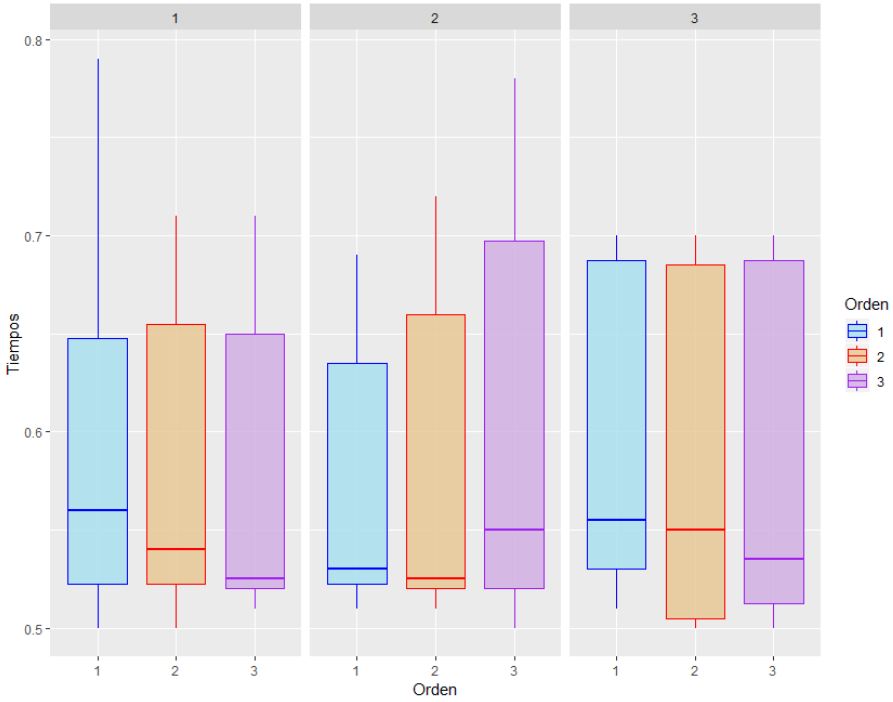

Figura 2: Tiempo total de ejecución dependiendo del orden del arreglo y la cantidad de núcleos para una proporción de 1/2.

Con una proporción de 1/2 2 podemos notar que las medias de los datos comparando con cada uno de los núcleos están más alejadas entre si y los tiempos de ejecución han disminuido. Al momento de evaluar el arreglo que contenía mitad números primos mitad no primo, la función estaba constantemente regresando los valores y no demoraba en los cálculos pues la cantidad estaba repartida de manera de numero estaba repartida uniformemente.

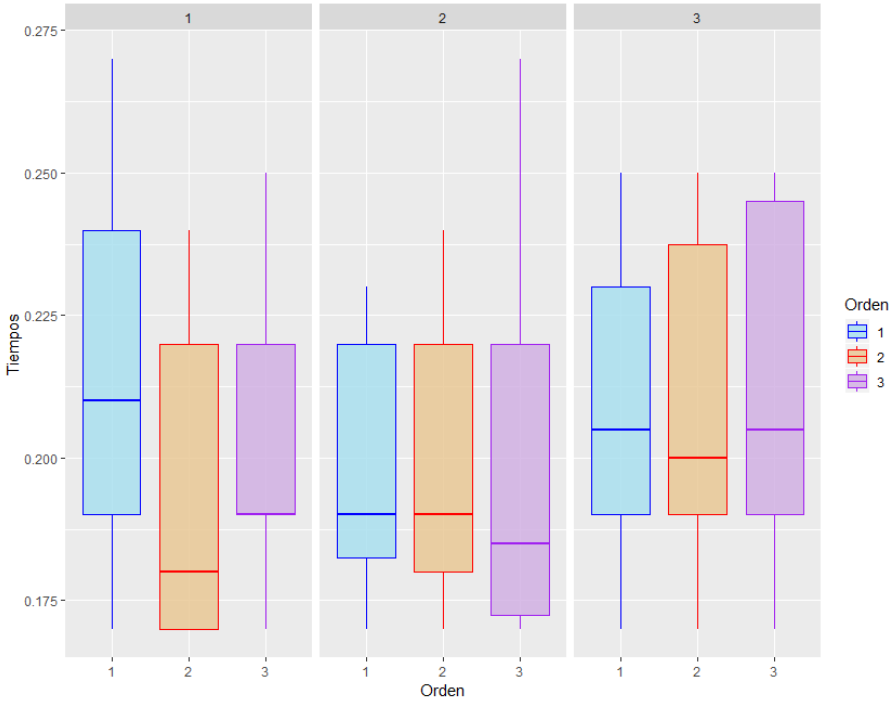

Figura 3: Tiempo total de ejecución dependiendo del orden del arreglo y la cantidad de núcleos para una proporción de 3/4.

En esta ultima gráfica 3 podemos observar que los tiempos de ejecución bajan aún más y se han alejado más las medias de los conjuntos de datos. Para la proporción 3/4 se observa que los resultados de las evaluaciones son más dinámicos que antes lo que se puede explicar a que existe una mayor cantidad de números primos dentro del arreglo evaluado.

Con esto se puede concluir que el numero de núcleos afecta de manera significativa en los tiempos de ejecución, pero hay que encontrar una cantidad adecuada de núcleos para usar para que los tiempos sean óptimos. Las proporciones de números primos y no primos dentro de los arreglos afecta a la hora del cálculo, que exista relleno en los arreglos puede ralentizar los tiempos así como que haya una cantidad mayor de números primos que pueden ser fácilmente deducibles, agiliza los tiempos. Por ultimo la cantidad que menos influye en los tiempos es el orden de los números en el arreglo pues se mantiene muy similar la media de los datos.

Referencias

[1] R: R Project, 2019
`https://www.r-project.org/`

[2] Satu Elisa Schaeffer: Práctica 2: autómata celular, 2019
`https://elisa.dyndns-web.com/teaching/comp/par/p2.html`

[3] Jorge Armando Serna Mendoza:Teoría de Colas, 2019
`https://sourceforge.net/projects/simulacionenr/files/P3/`