Teoría de colas

Simulación de sistemas

Marco Antonio Guajardo Vigil

11 de febrero de 2019

1. Introducción

La teoría de colas [4] es un área de las matemáticas que estudia el comportamiento de líneas de espera. Esto se ve a menudo cuando se trabaja con el cluster, ya que, retiene las tareas en una línea de espera hasta que terminen de realizarse todas. Esto a su vez resalta en el tiempo total de la ejecución.

El problema de ordenamiento de trabajos con la finalidad de minimizar el tiempo total de ejecución se llama *calendarización* (en inglés: scheduling) de tareas.

Para esta práctica se estudia el efecto del orden de ejecución para diferentes entradas. Se examina cómo las diferencias en los tiempos de ejecución de los diferentes ordenamientos cambian cuando se varía el **número de núcleos asignados** al cluster, utilizando como datos de entrada un vector que contiene numeros grandes, los cuales se obtienen de Primes [1] y no primos con un mismo número de dígitos. Se realizan estadísticas cómo pruebas para observar el efecto de la proporción de primos y no primos en el vector, igual como la magnitud de los números incluidos en él.

2. Implementación de R

Para la elaboración de este experimento, se hace uso de un software libre para computación estadística y gráficos llamado R [2], el cual nos permite realizar los cálculos necesarios para dicho experimento. Con él, se pueden controlar los datos estadísticos que se ocupan para dar seguimiento con la práctica, se necesita graficarlos para así poder compararlos mejor, ya que se maneja una cantidad de datos considerable y trabajaremos con ellos en forma estadística, por lo tanto, se recomienda el uso de este software ya que ayuda a paralelizar las acciones que sean necesarias, así se ahorra tiempo, haciéndolas simultáneamente.

3. Experimentación

Para llevar a cabo este experimento estadístico, como trabajo de ejemplo para tiempos de ejecución diversos para diferentes entradas, se utiliza la examinación de sí o no un número entero dado es un **número primo**, queriendo decir que no sea divisible entre ningún entero mayor a uno o menor a si mismo.

Se ultiliza el siguiente algotirmo:

```
primo <- function(n) { #Detectar primos
  if (n == 1 || n == 2) {
    return(TRUE)
  }
  if (n %%2 == 0) {
    return(FALSE)
  }
  return(TRUE)
}
```

Los números a tomar para la ejecución se almacenaron en un **dataframe**, para despues convertirse a un vector. El siguiente código se realizó con ayuda del reporte de Astrid [3]:

```
Primos <- read.table("primes1.txt", skip = 0) #Tomar numeros primos a partir de 6 digitos
Primos <- c(t(Primos))
```

Las variables que se usan para controlar el comportamiento de los núcleos, proporcionalidades, guardar los datos de los tiempos y la cantidad de números primos que se utilizan:

```
Cores <- seq(1, detectCores() - 1, 1)
Proporciones <- c(25, 50, 75) #Quantil
sumatoria <- data.frame()
NumP <- 500
```

Se contó con 3 núcleos para la realización del experimento. El experimento se repite 20 veces para cada configuración de núcleos y proporciones. Se asignan diferentes ordenes de realización de tareas para hallar la más óptima.

```
for (i in 1:rep) {
      ot <- c(ot, system.time(foreach(n = original, .combine=c) %dopar% primo(n))[3]) # de menor a
          mayor
      it <- c(it, system.time(foreach(n = invertido, .combine=c) %dopar% primo(n))[3]) # de mayor a
          menor
      at <- c(at, system.time(foreach(n = sample(original), .combine=c) %dopar% primo(n))[3]) # orden
          aleatorio
}
```

El algortimo que se usa para los números no primos es:

```
noPrimo <- function(n) { #Detectar no primos
  if (n == 1 || n == 2) {
    return(FALSE)
  }
  if (n %%2 == 0) {
    return(TRUE)
  }
  return(FALSE)
}
```

Los resultados obtenidos se grafican con ayuda de **ggplot**:

```
gg <- ggplot(sumatoria, aes(x = as.factor(Nucleos), y = value, group = interaction (Nucleos, Orden),
    fill = as.factor(Orden))) +
  geom_bar(stat="identity", position="dodge") + theme_gray(base_size = 14) + labs(x = "Cantidad de n\u{
      FA}cleos",
                                                                                  y = "Tiempo de ejecuci\u{F3}n en segundos",
                                                                                  fill = "Tipo de \n ordenamiento")
gg <- gg + scale_fill_manual( values = c("orange", "blue", "green"), aesthetics = "fill") +
  facet_grid(~Proporcion, scale="fixed") + theme(legend.title.align=0.5)
gg
```

4. Resultados

Tomando en cuenta el ordenamiento:

1. De menor a mayor.
2. De mayor a menor.
3. De forma aleatoria.

Se obtiene que las condiciones más óptimas, de acuerdo a la gráfica 1, se dan en la proporcionalidad uniforme de 0.5, al tomar dos y tres núcleos, el tiempo de ejecución fue el mínimo y con un gran parentezco en los dos casos, tomando aproximadamente 0.8 segundos en realizar el trabajo.

Donde se notó más inestabilidad y mayor tiempo de ejecución es cuando solo se llega a usar un núcleo para todos los casos, es lógico, ya que, mientras menos núcleos se utilicen, se llega a tener algunas restricciones al no usar todo el potencial que se tiene al usar más núcleos.

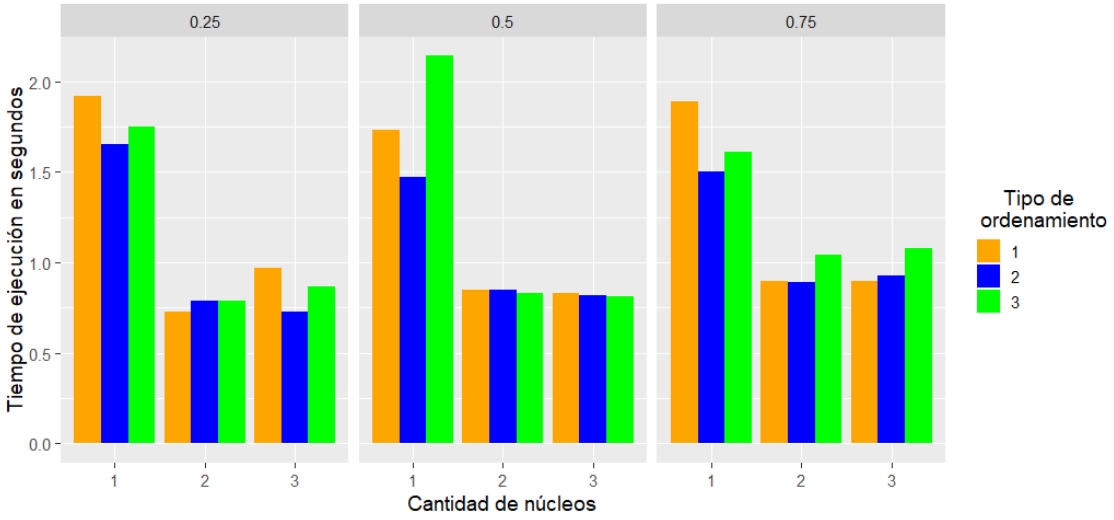

Figura 1: Gráfica del tiempo de ejecución y los núcleos usados, en relación a sus proporciones y ordenamientos.

5. Conclusión

Se esperó que el ordenamiento aleatorio (3) se comportase mejor que el resto en la mayoría de los casos, en la gráfica se aprecia que con núcleos mayor a uno, se vuelve el ordenamiento mas óptimo para la ejecución de estas tareas. Con núcleo uno actua de la peor manera, en especial, con la proporción de 0.5.

El ordenamiento de mayor a menor (2) es el que mejor se comporta cuando se realiza con un núcleo, siendo la mejor opción a considerar para este.

En el ordenamiento de menor a mayor (1) influye más que otros la proporcionalidad, ya que, en los casos donde se tienen tres núcleos, se aprecia que actua de mejor manera en una proporcionalidad más alta como en la de 0.75.

Las variables que controlan los núcleos y las proporcionalidades, son las que más llegan a influir en cualquier metodo de ordenamiento, siendo estas muy fundamentales para llegar a una buena optimización de las tareas. Intervienen en el ordenamiento de tal modo que pueden empeorarlos o mejorarlos.

Referencias

[1] The first fifty million primes. 2019. URL `https://primes.utm.edu/lists/small/millions/`.

[2] The R Project for Statistical Computing. 2019. URL `https://www.r-project.org/`.

[3] Astrid González. Práctica 3: Teoría de colas. pages 1–3, 2018. URL `https://sourceforge.net/projects/simulacion-de-sistemas/`.

[4] Satu Elisa Schaeffer. Práctica 3: Teoría de colas. 2019. URL `https://elisa.dyndns-web.com/teaching/comp/par/p3.html`.