Reporte Práctica 3

Teoría de Colas

Juan Pablo Chávez Granados

A 29 de Agosto de 2017

Teoría de colas

La teoría de colas es el estudio matemático de las colas o líneas de espera dentro de un sistema, esta teoría de estudia los tiempos de espera y la capacidad de realizar un trabajo sin llegar a colapsar. Estos factores tienen aplicaciones en diversas situaciones como los negocios, comercio, industria, ingenierías, transporte, logística y telecomunicaciones.

El fenómeno de colas tiene su origen en el desequilibrio temporal entre la demanda de un servicio y la capacidad del sistema para suministrarlo. Cuando hablamos de un fenómeno de colas se tiene que identificar los seis elementos básicos de los cuales se compone, estos elementos son:

1- La población: Es una fuente finita o infinita de información la cual llega a las colas.
2- La llegada: Esta es la forma en que llegan los clientes o datos a la instalación de servicio.
3- La cola: Ésta tiene dos parámetros que la forman; el número de colas y la longitud de la cola, donde el número de colas nos dice la existencia de una o múltiples colas, y la longitud de la cola nos habla de la capacidad para realizar el trabajo el cual puede ser infinito o limitado.
4- La selección: Ésta depende de una o más reglas que sirven para determinar el orden de servicio a los clientes.
5- El mecanismo de servicio: Consiste en una o varias instalaciones que realizan el servicio, cada una las instalaciones pueden tener uno o más canales paralelos (servidores).
6- La salida: Cuando el cliente ha sido atendido sale del sistema y pueden existir dos posibilidades, que regrese para un nuevo servicio o que no regrese para un nuevo servicio.[1,2]

Práctica

En la práctica de teoría de colas, se ha analizado la diferencia en tiempos de ejecución cuando se varía el número de núcleos asignados a una tarea.

- Código inicial

```
primo <- function(n) {
  if (n == 1 || n == 2) {
    return(TRUE)
  }
```

```
    if (n %% 2 == 0) {
        return(FALSE)
    }
    for (i in seq(3, max(3, ceiling(sqrt(n))), 2)) {
        if ((n %% i) == 0) {
            return(FALSE)
        }
    }
    return(TRUE)
}

desde <- 10
hasta <-  30
original <- desde:hasta
invertido <- hasta:desde
replicas <- 10
suppressMessages(library(doParallel))
registerDoParallel(makeCluster(detectCores()-1))
ot <-  numeric()
it <-  numeric()
at <-  numeric()
for (r in 1:replicas) {
    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
}
```

```
stopImplicitCluster()

summary(ot)

summary(it)

summary(at)
```

Este código busca los números primos en una secuencia que puede ir desde 1 a una cantidad finita de datos, en el ejemplo, se buscaron números primos que se encuentren entre 100 y 300, esta tarea se realiza en un cierto tiempo, éste puede variar con el número de réplicas, el orden y el número de clusters asignados para la tarea.

Posteriormente se modificó el código inicial y se utilizó el código de jbenavidesv87[3] como ejemplo del uso del "rbind" para la generación de matrices y la graficación de múltiples datos, también se utilizó el código de la práctica anterior para la graficación del "boxplot".

- Código Pablo

```
suppressMessages(library(doParallel))


desde <- 100

hasta <-  300

original <- desde:hasta

invertido <- hasta:desde

replicas <- 10

Tiempo <- data.frame()

Nucleos <- detectCores()


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

```
    for (i in seq(3, max(3, ceiling(sqrt(n))), 2)) {
        if ((n %% i) == 0) {
            return(FALSE)
        }
    }
    return(TRUE)
}


for (nucleo in 1:(Nucleos-1)){
registerDoParallel(makeCluster(nucleo))
ot <-  numeric()
it <-  numeric()
at <-  numeric()
for (r in 1:replicas) {
    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
}
stopImplicitCluster()
summary(ot)
summary(it)
summary(at)


  Tiempo <- rbind(Tiempo, c(ot, nucleo))
  Tiempo <- rbind(Tiempo, c(it, nucleo))
  Tiempo <- rbind(Tiempo, c(at, nucleo))
```

```
}

Tiempo[, "mean"] <- apply(Tiempo[, 1:10], 1, mean)
png("Tiempo.png")

plot(Tiempo[, 12], xlab = "Nucleos", ylab = "Tiempo (s)",
  main = NULL,
  ylim = c(min(Tiempo[, 12]), max(Tiempo[, 12])),
  xaxt='n'
  )
  axis(1, at=1:12, labels=c(
    "1o", "1i", "1a",
    "2o", "2i", "2a",
    "3o", "3i", "3a",
    "4o", "4i", "4a"
    ))
graphics.off()

boxplot(t(Tiempo[, 12]), xlab = "Nucleos", ylab= "Tiempo (s)", main = NULL,
ylim = c(min(Tiempo[, 12]), max(Tiempo[, 12])),
xaxt='n'
)
axis(1, at=1:12, labels=c(
    "1o", "1i", "1a",
    "2o", "2i", "2a",
    "3o", "3i", "3a",
    "4o", "4i", "4a"
```

```
  ))
png("boxplot.png")
graphics.off
```

En este código se conservó activo el paquete “doParallel”, el cual permite hacer múltiples cálculos haciendo uso de los núcleos en la computadora, también permite dar múltiples ordenamientos para el uso de esos núcleos. En el código se generó un vector “Tiempo”, al cual se le asignó un data.frame para guardar los tiempos promedio de las tareas realizadas, una variable “Nucleos” que sirve para formar la función cluster, ésta sirve para dar el números de núcleos que se utilizarán en la tarea.

Resultados

Como resultados se obtuvo el tiempo promedio de la tarea, la cual fue buscar los números primos en una secuencia de 100 a 300 y se graficó en un boxplot. En la figura 1 se observa el promedio del tiempo para los diferentes órdenes, y como éste disminuye cuando se pasa de un orden original, un orden invertido y orden aleatorio.

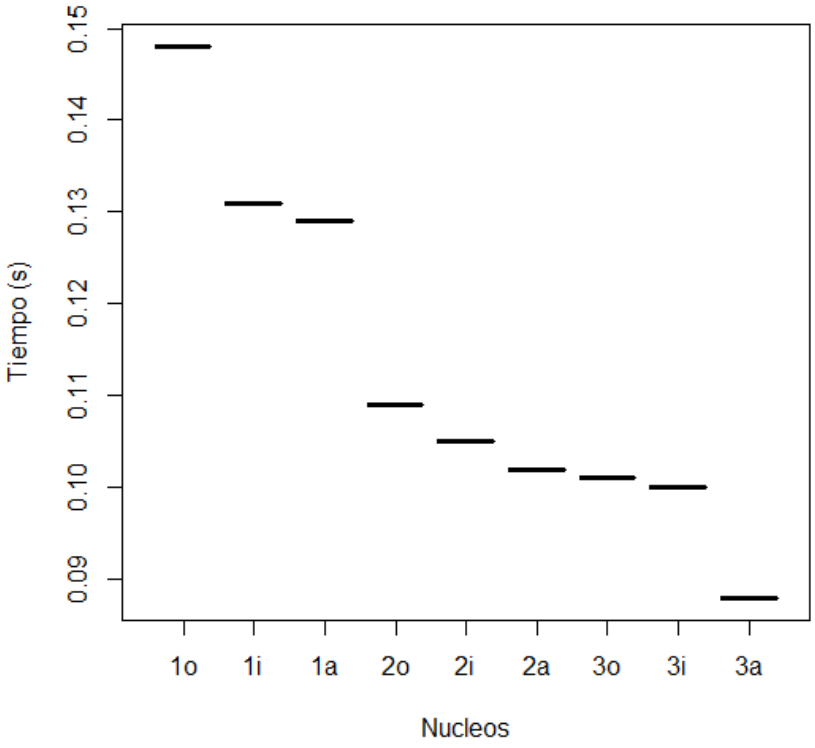
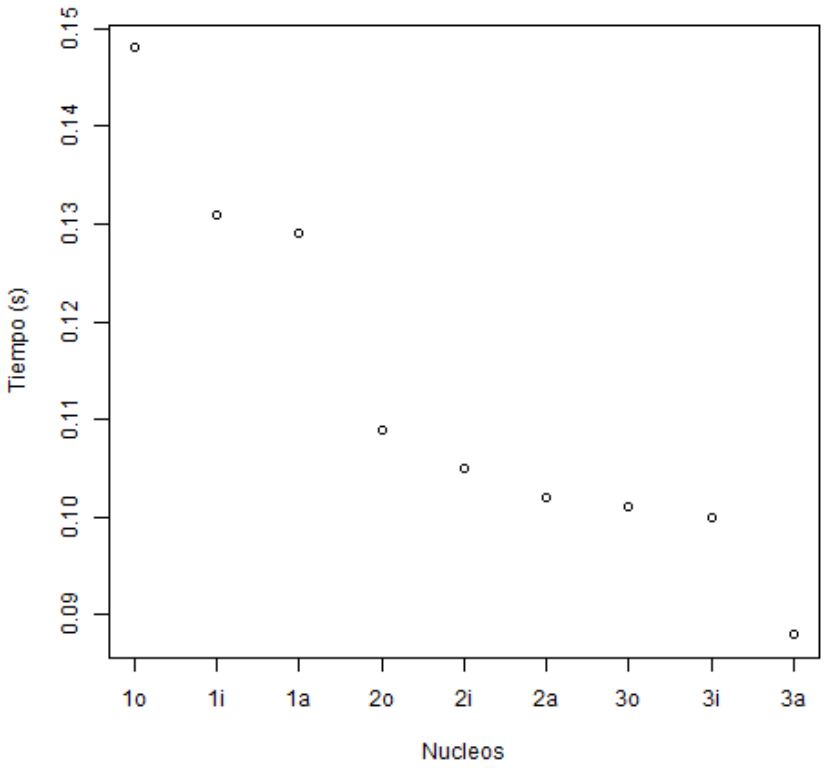


Fig. 1. Gráfica caja bigotes de los tiempos promedios para los distintos ordenamientos.

También se puede observar en la gráfica caja de bigotes como de manera gráfica nos expresa que los tiempos mínimos y máximos son muy parecidos al promedio para realizar la tarea.

En la figura 2, la cual es una gráfica de puntos del tiempo promedio se puede visualizar como cambia la densidad del tiempo promedio y similar a lo obtenido en la gráfica caja bigotes se ve una disminución en los tiempos de ejecución para los ordenamientos, se observa que para obtener menores tiempos de ejecución de una tarea es mejor utilizar ordenes aleatorios e incrementar el número de núcleos que realizarán dicha tarea.



Conclusiones

De esta práctica se puede concluir que la teoría de colas se utiliza en la vida cotidiana y el uso de este modelo matemático facilitaría la realización de varias tareas, además se puede concluir que el uso de múltiples núcleos (servidores) si disminuye el tiempo de ejecución de una tarea, pero a su vez el tiempo de ejecución es afectado por el ordenamiento que se le da a la cola, tal como lo dice la literatura los seis aspectos que influyen en el modelo de colas intervienen explícitamente en la ejecución de la

tarea, la selección de una cola es el aspecto que influye más y es en el cual podemos ver que se seleccionan números primos, esto se encuentra la función primo la cual es la regla de selección para ejecutar el trabajo, al tener un orden aleatorio se tiene una probabilidad de 0.5 de obtener un número primo así como de no tenerlo. Si se siguiera un orden original la selección de un número primo será más tardada así como en el orden invertido. La última conclusión es sobre la optimización del código al dejar solo los ordenamientos aleatorios los cuales disminuyen los tiempos de ejecución de tareas en un orden magnitud.

Bibliografía

1) de la Fuente García, David, Pino Díez, Raúl, (2001), *Teoría de líneas de espera: modelos de colas*, Ed. Univ. de Oviedo.
2) Sarabia Viejo, Angel, (1996), *La investigación operativa: una herramienta para la adopción de decisiones*, Ed. Univ. Pontifica Comillas.
3) jbenavidesv87, Codigo de Github.