Teoría de Colas

Abraham Azael Morales Juárez 1422745

12 de febrero de 2019

1. Introducción

La teoría de colas, es el estudio matemático de las características de sistemas de líneas o colas. Estas se presentan diariamente en la vida cotidiana. Esto ocurre inclusive, por ejemplo, cuando realizamos una llamada, en ocasiones tenemos que esperar porque el usuario está ocupado. Esto también ocurre en el supermercado, almacenes, aeropuertos, solo por mencionar algunos. Por esta razón es gran importancia realizar estudios en el comportamiento de estos tipos de sistemas [1].

2. Objetivos

Examinar cómo las diferencias en los tiempos de ejecución de los diferentes ordenamientos cambian cuando se varía el número de núcleos asignados al cluster. Usar datos de entrada de un vector que contiene primos grandes y no-primos con un mismo número de dígitos. Investiga también el efecto de la proporción de primos y no primos en el vector. Además el efecto de la magnitud de los números incluidos en el vector, con pruebas estadísticas adecuadas [3].

3. Resultados y Discusión

En los datos obtenidos de la simulación se puede observar claramente la gran diferencia de procesamiento de datos, debido a la cantidad de núcleos destinados para trabajar con la información. También como era de esperarse, la cantidad de información influye en el tiempo que tardó en realizarse el experimento. Se realizó un análisis en paralelo y las especificaciones utilizados para la realización de esta práctica son las siguientes: 1 núcleo, 6 núcleos, y 7 núcleos, para los cuales los rangos de números fueron: 10,000 – 50,000, 50,000 – 250,000 y 250,000 – 500,000. Cabe destacar que se usaron 2 procesadores distintos core i7 y core i5, 8 y 2 núcleos respectivamente. Se usaron distintos códigos los cuales definían funciones, tanto para números primos como para compuestos, que son los siguientes códigos: EL código para obtener la información y almacenarla en vectores es el siguiente:

```
compuesto <- function(n) {
  if (n < 4) {
    return(FALSE)
  }
  for (i in 2:(n-1)) {
    if (n %% i == 0) {
      return(TRUE)
    }
  }
  return(FALSE)
}

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)
}
```

Figura 1: Código para designar la función primo y compuesto

En la figura 3, que son gráficas caja-bigote, se presentan los resultados obtenidos del análisis con 7 núcleos, en los cuales las mejores gráficas se obtienen con los rangos menores, es decir, 10,000 – 50,000, en esta gráfica se observa que se obtuvo los primeros

```
suppressMessages(library(doParallel))
registerDoParallel(makeCluster(detectCores(7) - 1))
otp <-  numeric()
itp <-  numeric()
atp <-  numeric()
for (r in 1:replicas) {
  otp <- c(otp, system.time(foreach(n = original, .combine=c) %dopar% primo(n))[3]) # de menor a mayor
  itp <- c(itp, system.time(foreach(n = invertido, .combine=c) %dopar% primo(n))[3]) # de mayor a menor
  atp <- c(atp, system.time(foreach(n = sample(original), .combine=c) %dopar% primo(n))[3]) # orden aleatorio
}
stopImplicitCluster()

summary(otp) # primos desde:hasta
summary(itp) # primos hasta:desde
summary(atp) # primos aleatorio

suppressMessages(library(doParallel))
registerDoParallel(makeCluster(detectCores(7) - 1))
otc <-  numeric()
itc <-  numeric()
atc <-  numeric()
for (r in 1:replicas) {
  otc <- c(otc, system.time(foreach(n = original, .combine=c) %dopar% compuesto(n))[3]) # de menor a mayor
  itc <- c(itc, system.time(foreach(n = invertido, .combine=c) %dopar% compuesto(n))[3]) # de mayor a menor
  atc <- c(atc, system.time(foreach(n = sample(original), .combine=c) %dopar% compuesto(n))[3]) # orden aleatorio
}
stopImplicitCluster()

summary(otc) # compuestos desde:hasta
summary(itc) # compuestos hasta:desde
summary(atc) # compuestos aleatorio
# graficas boxplot
par (mfrow=c(1,1))
nombres <- c("Original", "Invertido", "Aleatorio")
boxplot(summary(otp), summary(itp), summary(atp), main="Números primos 6 núcleos", xlab="Vectores" , ylab="Tiempo", boxwex=0.15, names = nombres, range = 0)
boxplot(summary(otc), summary(itc), summary(atc), main="Números compuestos 6 núcleos", xlab="Vectores" , ylab="Tiempo", boxwex=0.15, names = nombres, range = 0)
```

Figura 2: Código vector, boxplot y paralelo

resultados casi al mismo tiempo, pero teniendo datos muy irregulares con bigotes muy largos y en donde particularmente con los números compuestos, se observa más claramente la priorización del análisis debido a que, realizar todos las acciones que se necesitan partiendo de datos grandes es más complicada que partir de datos más sencillos. Aunque, se puede observar la diferencia que hay entre el análisis de 7 núcleos y de 1 núcleo. Sin embargo, el análisis con el de 6 núcleos, no hubo mucha variación con el de 1, a diferencia de lo que se ha reportado [4], lo más probable fue por el hecho que las computadoras usadas tienen designaciones prioritarias que pueden requerir el uso de núcleos para actualizarse o simplemente ejecutarse en segundo plano [2].

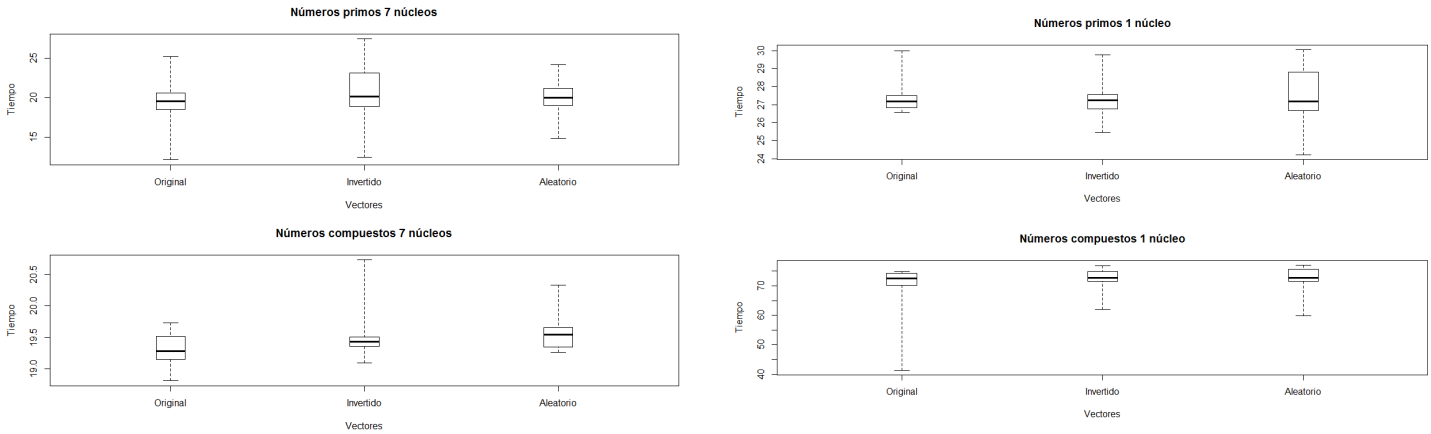
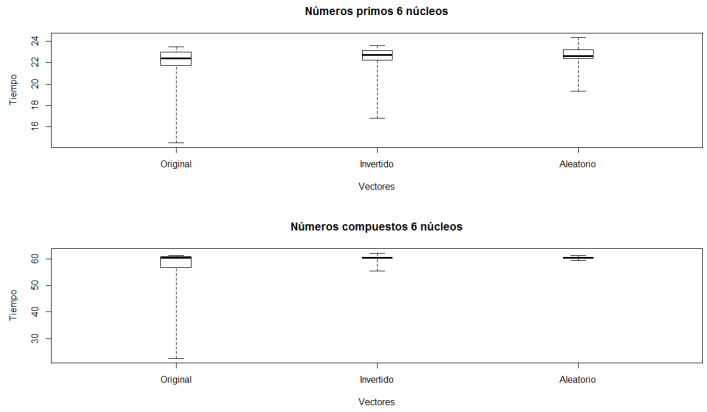


Figura 3: 7, 6 y 1 núcleo, 10,000 - 50,000



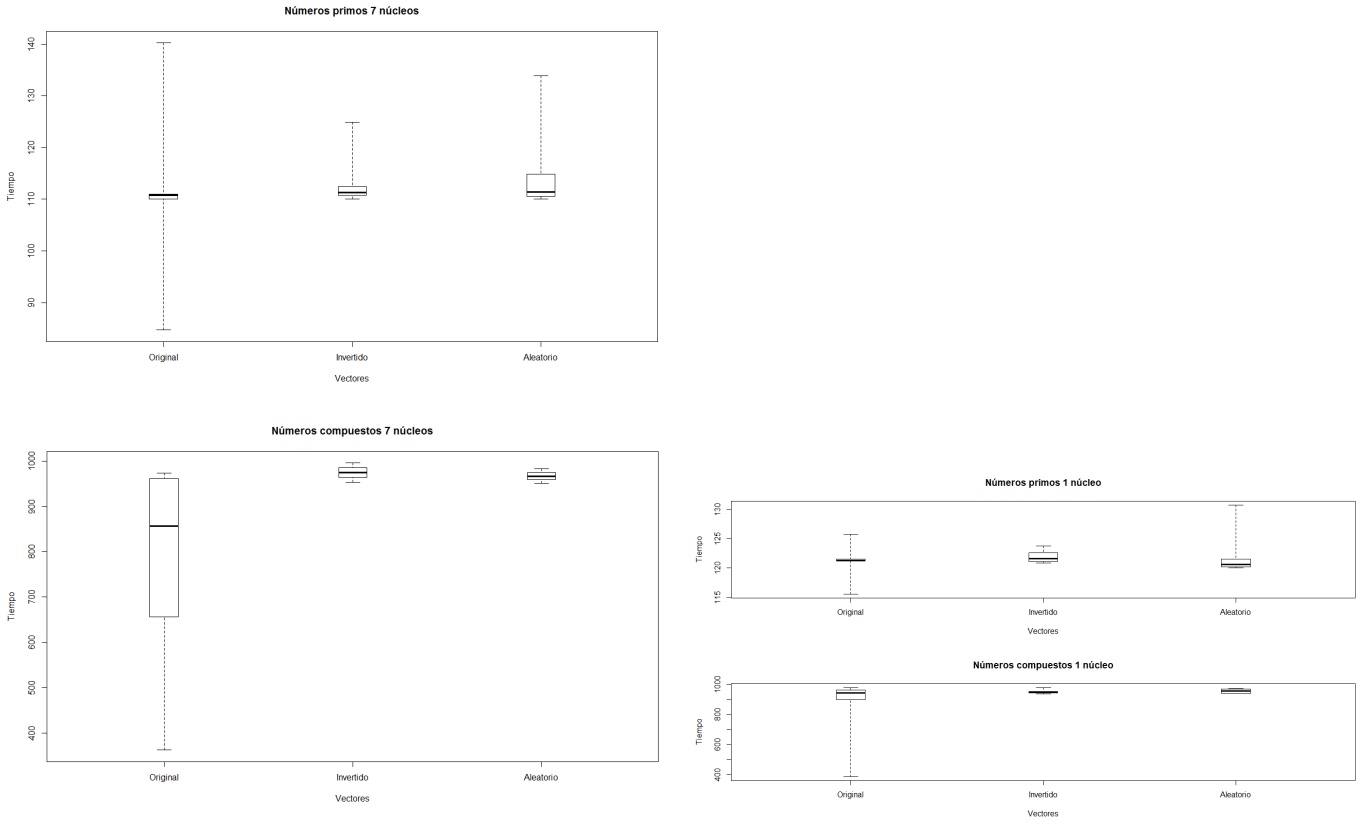
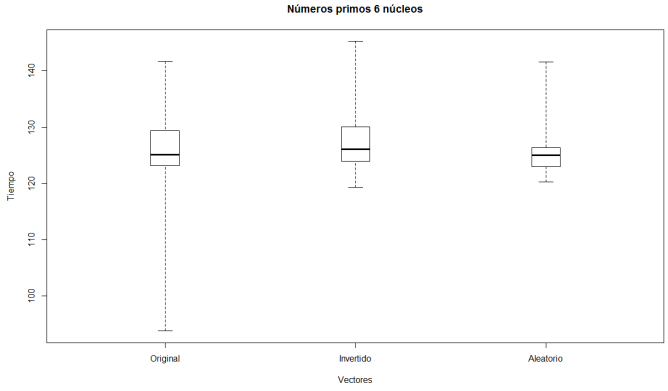


Figura 4: 7, 6 y 1 Núcleo 50,000 - 250,000



Con los resultados de 50000 – 250000, igual que los anteriores son muy distintos a los esperados y reportados [4], los tiempos de análisis son más cortos, (al menos con los números primos), con la computadora que tiene el procesador core i5. Mientras que con los 6 y 7 núcleos fue un poco mayor el tiempo esperado, pero similar entre ellos.

En el análisis más grande que se realizó, no se pudo obtener con 6 núcleos, por el tiempo de procesamiento que cada corrida requería. Aquí si se observó una diferencia muy grande entre los procesadores core i7 y el core i5, específicamente con los números compuestos, como se esperaba., y un poco menos con los números primos. Aunque, los resultados obtenidos siguen viéndose afectados, principalmente por los sistemas operativos instalados, que priorizan funciones de Windows ocasionando que los núcleos hagan otras tareas además de las asignadas. Afectando el proceso en paralelo que se está ejecutando [2].

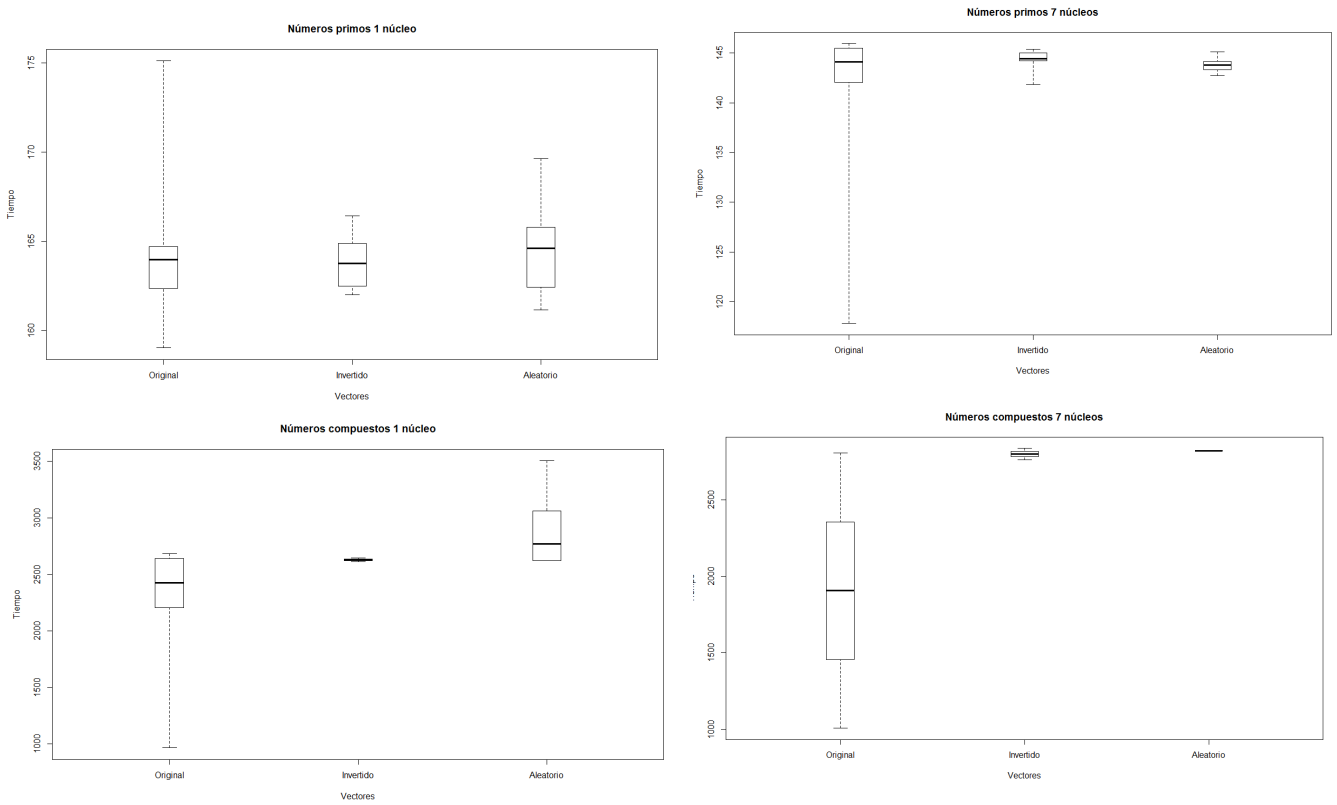


Figura 5: 7 y 1 núcleo, 250,000 - 500,000

4. Conclusiones

Estos análisis son de gran importancia, debido a que nos explican el comportamiento de las líneas de espera y el como poder tratar con ellas, con las herramientas disponibles. El uso de la herramienta paralelo es de gran importancia, debido a que con estos se puede llegar a predecir el comportamiento de distintos fenómenos o sistemas. Los equipos que se utilicen para realizar este tipo de simulaciones necesitan tener un control total sobre el software, para poder manipular y priorizar de manera voluntaria los núcleos.

Referencias

[1] Pedro García. Aplicando la teoría de colas en dirección de operaciones. *Universidad Politécnica de Valencia*, 2015.

[2] Rodríguez D. Joaquín López. Análisis de rendimiento de algoritmos paralelos. *Tecnológico de Costa Rica*, 2014.

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

[4] Javier Serrano. Memoria, gestión de procesos en los sistemas operativos. *TFC: Arquitectura de computadoras y sistemas operativos*, 2011.