Teoría de colas

Francisco Gerardo Meza Fierro

1. Introducción

En esta práctica se estudiará el efecto del orden de ejecución de trabajos y el número de núcleos utilizados en esa medida. Para ello, se tomará una lista de números, el programa seleccionará aquellos que sean primos y los ordenará mediante tres métodos distintos con objetivo de comparar los mencionados tiempos de ejecución.

2. Número de núcleos asignados

El código original fija un único núcleo para correr el programa el número de veces que el usuario desee, fijando además la lista de números de los cuales, mediante la función `primo`, se tomarán aquellos que sean primos y se ordenarán de manera ascendente, descendente y aleatoria, registrando sus tiempos en las variables `ot`, `it` y `at` respectivamente.

Para llevar registro de la cantidad de núcleos que se asignan en la ejecución, se agregó al inicio la función `for` permitiendo repetir el código igual al número de núcleos que posea el equipo que ejecute el programa. La lista de números, para esta práctica, inició en mil y terminó en tres mil. Durante ese proceso se registraron todos los datos, se promediaron y se ordenaron por tipo de ordenamiento y por cantidad de núcleos empleados, obteniendo el siguiente cuadro:

Cuadro 1: Tiempos promedio en segundos

	ot	it	at
1 núcleos	1.599	1.581	1.589
2 núcleos	1.065	1.075	1.052
3 núcleos	1.09	1.058	1.039
4 núcleos	1.053	1.04	1.036

Como era de esperarse, mientras mayor sea el número de núcleos designados para ejecutar el programa, menor será el tiempo de ejecución del mismo. Esto podría explicarse de una manera más sencilla si vemos la cantidad de carriles de una carretera como los núcleos designados al programa y a los vehículos como la cantidad de datos con los que el programa trabajará: claramente, si se excluyen los altos, las luces rojas de los semáforos y la poca cultura vial de las personas, es claro que mientras más carriles tenga una carretera, menor será la probabilidad de que los vehículos se aglomeren provocando tráfico; en cambio si la carretera tuviese uno o muy pocos carriles, es más probable que los vehículos tarden más tiempo en avanzar por esa carretera. Lo mismo es de esperarse al emplear más núcleos para la ejecución del programa, o al menos es lo que se esperaría. La figura 1 muestra con mayor comodidad visual los datos del cuadro 1.

Algo interesante de notar es que, en cualquiera de los cuatro casos en que se usan núcleos distintos, de los tres ordenamientos distintos, aunque sea muy poca la diferencia, el menos eficiente sería el ordenarlos de manera ascendente mientras que el más eficiente parecería ser el ordenarlos de manera aleatoria. Gracias a esa poca diferencia visible que existe, se podría suponer que no hubiera diferencia entre usar un método de ordenamiento en específico.

Es evidente que usar un núcleo es mucho menos eficiente que usar dos o más. Sin embargo, con ayuda de esa misma gráfica, se ve que la diferencia de tiempos usando dos, tres y cuatro núcleos no es tanta como la que se esperaba. A pesar de eso, debido a la gran diferencia que hay entre el usar un núcleo a usar dos o más, se podría suponer que es muy influyente el asignar cierto número de núcleos para este programa.

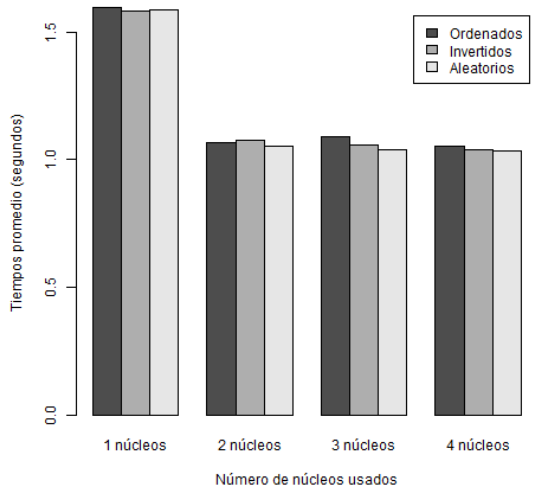

Figura 1: Gráfico del cuadro 1

3. Pruebas estadísticas

Podrían hacerse mucha suposiciones e intuiciones sobre qué método es mejor y sobre cuántos núcleos habrá que asignar para correr el programa de una mejor manera, pero hacerlo basándose en los promedios obtenidos (reflejados en el cuadro 1) o en la gráfica de barras de esos promedios (reflejados en la figura 1) no sustentaría nada mas que una simple suposición. De manera que, para asegurar si alguno de estos tres métodos de ordenamiento es mejor o si usar cierta cantidad de núcleos es mejor que usar menos o más, se realizó una prueba estadística a los datos obtenidos.

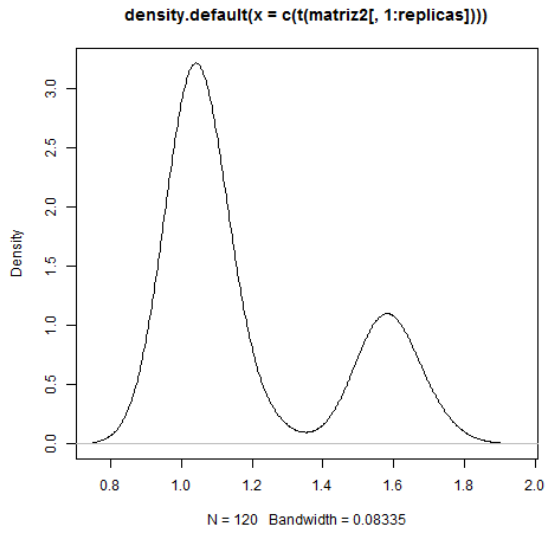

Figura 2: Curva de densidad de los datos

La figura 2 se obtuvo mediante la función `density` y muestra claramente que los datos obtenidos no siguen una distribución normal, por tanto para analizar los datos es necesario emplear una prueba no paramétrica para analizar los datos.

Se decidió emplear la prueba de Kruskal-Wallis para llevar a cabo dicho análisis ya que ésta determina si existe diferencia significativa entre dos grupos de datos y, básicamente el problema en esta práctica se resume a comparar dos grupos de datos:

1. Ver si hay diferencia en ordenar los datos dependiendo el tipo de ordenamiento.

2. Ver si hay diferencia en emplear cierta cantidad de núcleo para ordenar los datos.

La prueba de Kruskal-Wallis toma como hipótesis nula que los grupos de datos son estadísticamente iguales. Si al aplicar la prueba se obtiene un $p-$valor menor o igual a 0,05, se rechazaría la hipótesis nula y eso significa que los grupos de datos no son iguales; en cambio, si se obtiene un $p-$valor mayor a 0,05 se acepta la hipótesis nula.

Al aplicar la función `kruskal.test` en ambos grupos se obtuvieron los siguientes $p-$valores:

Cuadro 2: $p-$valores

	núcleos	ordenamiento
p-valor	1.33E-15	0.7541

Con esos valores finalmente se pueden hacer dos afirmaciones:

1. Existe una diferencia significativa en el número de núcleos que se asignan, por lo que la idea que se tenía previamente en el último párrafo de la segunda sección de esta práctica resultó acertada: usar un núcleo es mucho menos eficiente que usar dos o más.

2. No existe una diferencia significativa en el uso específico de alguno de los tres tipos de ordenamiento, por lo que la idea que se tenía en el penúltimo párrafo de la segunda sección de esta práctica fue acertada: no hay diferencia entre usar un método de ordenamiento en específico.