Tarea 3 Simulación de Teoría de Colas

Susana Ruiz Nuñez 2032426

1. Planteamiento de la Teoría de colas

Se requiere estudiar el efecto del orden de ejecución de trabajos y el número de núcleos utilizados en la teoría de colas [3]. La teoría de colas estudia el comportamiento de líneas de espera. Los trabajos que están esperando ejecución en un cluster esencialmente forman una línea de espera. Como medidas de interés principal que ayudan a caracterizar el comportamiento de una línea de espera incluyen el tiempo total de ejecución.

2. Metodología

Se utiliza, en este caso, 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 comienza con un listado de números primos descargados de la web [1] y el análisis de si son o no primos con un algoritmo sencillo [3]. Ya obtenidos la lista de números primos y el análisis para saber si lo son o no, es necesario crear una serie de trabajos fáciles o sea, números que no sean primos que el programa pueda resolver rápidamente.

Se creó un for para generar una lista de valores que estuvieran entre dos primos grandes; si se tienen dos primos consecutivos, los valores que se encuentran entre ellos no lo son. Estos valores se concatenan junto a los números primos en una lista total que será con la que se seguirá trabajando a continuación.

```
facil = []
for cand in range(minimo+1, maximo-1):
        facil.append(cand)
total = primos + facil
```

Posteriormente era necesario buscar un mecanismo para poder variar el número de núcleos asignados al cluster. El equipo donde se está trabajando el proyecto cuenta con AMD Ryzen 3 3200U with Radeon Vega Mobile Gfx (4 CPUs), 2.6GHz, dos núcleos físicos y dos virtuales. Para la variación de los núcleos se utilizaron los paquetes psutil y multiprocessing.

3. Pruebas Estadísticas

Se realizaron pruebas estadísticas [2] para ver la relación existente entre los valores primos obtenidos de la web y los valores no primos generados.

3.1. Coeficiente de correlación de Pearson

Se utiliza para comprobar que las dos muestras de valores que estamos tratando estén relacionadas. Prueba la relación lineal entre dos conjuntos de valores.

Supuestos

- Las observaciones en cada muestra son independientes y están distribuidas de manera idéntica (iid).
- Las observaciones de cada muestra se distribuyen normalmente.
- Las observaciones en cada muestra tienen la misma varianza.

Interpretación

- H0: las dos muestras son independientes.
- H1: existe una dependencia entre las muestras.

Resultados: Como el coeficiente de pearson $_p > 0.05$ entonces se plantea que son independientes.

```
stat=0.420, p=0.580
Probably independent
```

3.2. Prueba t de Student

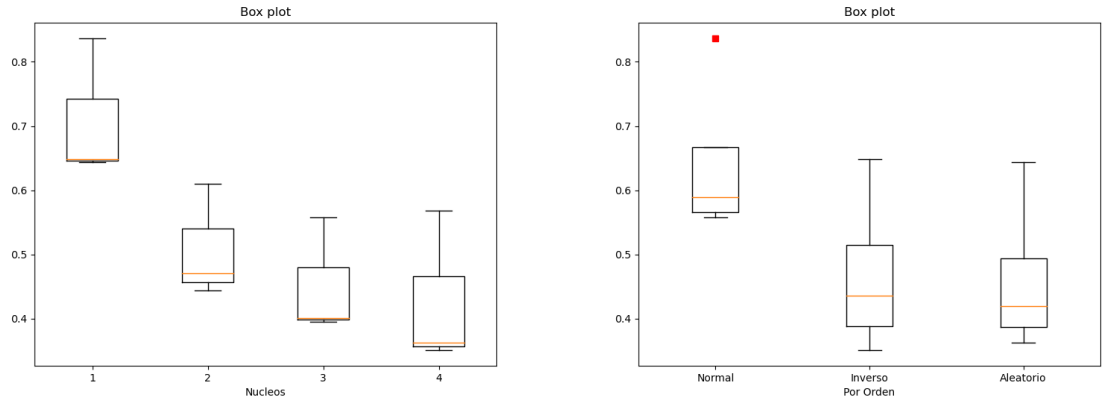
Comprueba si las medias de las dos muestras independientes son significativamente diferentes.

Interpretación

- H0: las medias de las muestras son iguales.
- H1: las medias de las muestras son desiguales.

Resultados: Como el coeficiente de pearson $_p > 0.05$ entonces se plantea que tienen la misma distribución.

```
stat=1.452, p=0.242
Probably the same distribution
```



(a) Utilización de núcleos.

(b) Orden de los datos.

Figura 1: Resultados por utilización de núcleos y orden de los datos.

4. Resultados

Como se muestra en la (Figura 1) hay una diferencia considerable en los tiempos de ejecución, se muestra mucho más al aumentar los núcleos que en los diferentes tratamientos de los datos, se trataron de manera Normal, es decir, en el orden que aparecen en la lista, luego de manera Inversa y finalmente aleatoriamente. En el caso del uso de los núcleos se muestra como se acercan a 0 mientras más núcleos se utilizan. Se da una muestra de los datos obtenidos con el uso de Python3.7, para un núcleo y para cuando se utilizaron los 4 núcleos.

```
DescribeResult(nobs=10, minmax=(0.49785828590393066, mean=0.8369509935379028)
DescribeResult(nobs=10, minmax=(0.500037670135498, mean=0.6482812881469726)
DescribeResult(nobs=10, minmax=(0.5040359497070312, mean=0.6433071374893189)

...

DescribeResult(nobs=10, minmax=(0.32102465629577637, mean=0.5686076641082763)
DescribeResult(nobs=10, minmax=(0.32302260398864746, mean=0.3506206512451172)
DescribeResult(nobs=10, minmax=(0.3180241584777832, mean=0.3628715753555298)
```

En la (Figura 2) se muestra el comportamiento por el Ordenamiento a que fueron tratados los datos. Se nota a simple vista que cuando el Orden tenía los números primos delante era más complejo el análisis de los datos. Lo que cambia al invertir el orden y tratar los valores fáciles primero y de manera aleatoria.

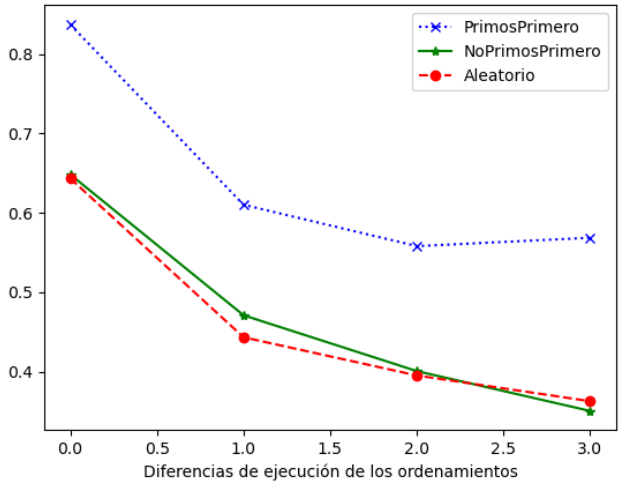


Figura 2: Ejecución por Orden

5. Conclusiones

Se puede concluir con los experimentos realizados que al aumentar el número de núcleos asignados al cluster, se realiza un mejor trabajo de procesamiento observándose gran diferencia en los tiempos de procesamiento. Lo mismo ocurre en el ordenamiento de los datos, si se juega al tetris con las fichas más difíciles al inicio se hará más complejo el trabajo a la pc.

Referencias

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

[2] Jason Brownlee. 17 statistical hypothesis tests in python (cheat sheet), 2018. URL `https://machinelearningmastery.com/statistical-hypothesis-tests-in-python-cheat-sheet/`.

[3] Elisa Schaeffer. Práctica 3: teoría de colas, 2020. URL `https://elisa.dyndns-web.com/teaching/comp/par/p3.html`.