Práctica 3: Teoría de colas

Raul L.

1 de marzo de 2022

1. Introducción

La teoría de colas es un área de las matemáticas que estudia el comportamiento de líneas de espera. Los trabajos que están esperando ejecución en un cluster esencialmente forman una línea de espera. Medidas de interés que ayudan caracterizar el comportamiento de una línea de espera incluye, el tiempo total de ejecución. En esta práctica se estudiará el efecto del orden de ejecución de trabajos y el número de núcleos utilizados en esta medida[1].

2. Objetivo

Examinar las diferencias en los tiempos de ejecución variando algunos o todos de los siguientes aspectos:

El orden de los números.

La cantidad de núcleos asignados al cluster.

La variante de la rutina para determinar si un número es primo
Aplicando pruebas estadísticas adecuadas y visualización científica clara e informativa. [1].

3. Código

Con el siguiente código se examinó como las diferencias en los tiempos de ejecución de los diferentes ordenamientos cambian cuando se varía el número de núcleos asignados al clúster, utilizando como datos de entrada diferentes procesos para obtener primos grandes. Además el programa hace un análisis estadístico para determinar si tienen una relación significativa o no, asímismo, este se graficó en barras horizontales .
El código base se sacó del repositorio de Elisa Schaeffer.

Código en Phyton

`https://github.com/satuelisa/Simulation/blob/master/QueuingTheory/fixedshuffle.py`

Código creado en Python

`https://github.com/Raullr28/Resultados/tree/main/P3`

Iniciamos definiendo dos formas de encontrar números, teniendo el método para encontrar números primos se establecen los 3 parámetros de recorrido, las repeticiones y los núcleos presentes.

```python
from math import ceil, sqrt
def primo1(n):# algoritmo para encontrar los numeros primos
     if n < 3:
        return True
     for i in range(2, n):
        if n % i == 0:
            return False
     return True


def primo2(n1):
     if n1 < 4:
        return True
     if n1 % 2 == 0:
       return False
     for i in range(3, n1 - 1, 2):
        if n1 % i == 0:
            return False
     return True
```

Figura 1: Recorte del código de Raul L. del código de Python `https://github.com/Raullr28/Resultados/tree/main/P3`

Continuamos con un ciclo donde entran en juego los parámetros anteriormente dados, se hace un recorrido con el primer método para encontrar números primos utilizando los 3 parámetros de recorrido, al igual que con el segundo método para encontrar números primos lo cual genera un ciclo completo. El ciclo se repite dependiendo el numero de núcleos de cada computadora.

```python
tiempos = {"ot1": [], "it1": [], "at1": []}
tiempos2 = {"ot2": [], "it2": [], "at2": []}
for x in range(1, cores-1):
    with multiprocessing.Pool(processes = x ) as pool:
        for r in range(replicas):
            t = time()
            pool.map(primo1, original)
            tiempos["ot1"].append(time() - t)
            t = time()
            pool.map(primo1, invertido)
            tiempos["it1"].append(time() - t)
            t = time()
            pool.map(primo1, aleatorio)
            tiempos["at1"].append(time() - t)
            t = time()
            pool.map(primo2, original)
            tiempos2["ot2"].append(time() - t)
            t = time()
            pool.map(primo2, invertido)
            tiempos2["it2"].append(time() - t)
            t = time()
            pool.map(primo2, aleatorio)
            tiempos2["at2"].append(time() - t)
    for tipo in tiempos:
        print('Con la cantidad de núcleos de: ', x)
        print(describe(tiempos[tipo]),tipo)
        print('')
    for tipo in tiempos2:
        print('Con la cantidad de núcleos de: ', x)
        print(describe(tiempos2[tipo]),tipo)
        print('')
```

Figura 2: Recorte del código de Raul L. del código de Python `https://github.com/Raullr28/Resultados/tree/main/P3`

Se utilizó un sistema estadístico para relacionar si existía un una dependencia entre los valores dados en el ciclo.

```
stat, p = pearsonr(tiempos["it1"],tiempos2["it2"])
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
        print('Probably independent')
else:
    print('Probably dependent')

stat, p = pearsonr(tiempos["at1"],tiempos2["ot2"])
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
        print('Probably independent')
else:
    print('Probably dependent')

stat, p = pearsonr(tiempos["ot1"],tiempos["at1"])
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
        print('Probably independent')
else:
    print('Probably dependent')
```

Figura 3: Recorte del código de Raul L. del código de Python `https://github.com/Raullr28/Resultados/tree/main/P3`

4. Resultados

En la figura se muestra el tiempo que transcurrió el código al procesar las diferentes formas en encontrar los números primos, para ello, se varió el número de núcleos.

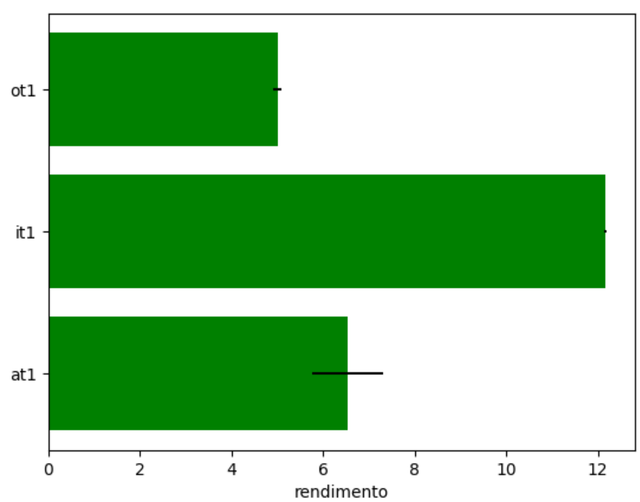

Figura 4: Gráfica tomada del repositorio de Raul L. del código de Python `https://github.com/Raullr28/Resultados/blob/main/P3/Figure_1.png`

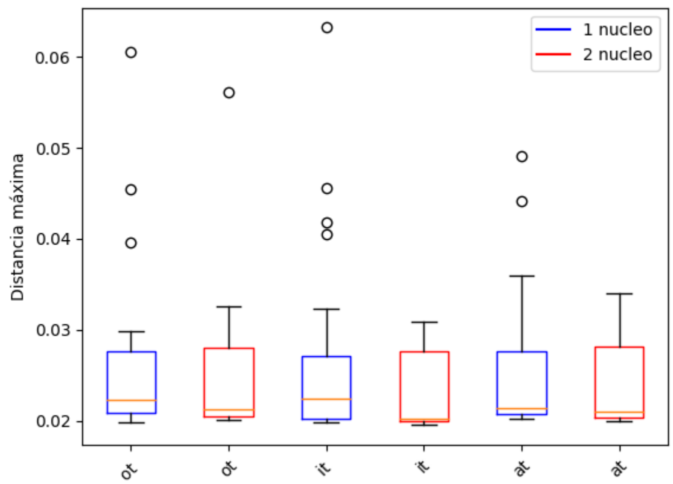

Figura 5: Gráfica tomada del repositorio de Raul L. del código de Python https://github.com/Raullr28/Resultados/blob/main/P3/Figure_2.png

5. Conclusión

Se creó un código que puede variar el número de núcleos para la correcta realización de la práctica 3, así como también se analizaron diferentes formas de en encontrar los números primos, para ello, se midió el tiempo en cada procedimiento, utilizando un análisis estadístico, con el cual se obtuvo una relación entre los números de núcleos y el tiempo realizado por cada proceso, lo cual se puede observar en la gráfica de lineas.

Referencias

[1] E. Schaeffer. Práctica 3: teoría de colas. 2022. URL https://satuelisa.github.io/simulation/p3.html.