Análisis de la Complejidad de la Solución Recursiva del Problema de las Torres de Hanoi con Diferentes Niveles de Poder de Cómputo

Arturo Márquez Flores

1. Introducción

El análisis de algoritmos cobra su importancia en el hecho de que las computadoras podrán tener niveles de rapidez y de memoria cada vez más grandes pero que nunca serán infinitos. Por tal razón, es importante considerar algoritmos que sean eficientes en su manejo de los recursos computacionales. Además, computadores con diferentes niveles de poder de cómputo podrían tener diferentes tiempos de terminación de un algoritmo, pero, en general, el beneficio obtenido de eficientar un algoritmo es mejor que el de tener más poder de cómputo. Asintóticamente, es posible que una computadora menos poderosa que otra pueda tardarse menos en resolver un problema si el algoritmo que utiliza es más eficiente que el que utiliza la computadora más poderosa [1]. Sin embargo, si la implementación del algoritmo es la misma, el poder de cómputo si puede llegar a marcar diferencias, aunque muchas veces sean sutiles. En este pequeño proyecto utilizamos la solución recursiva al problema de las torres de Hanoi para analizar las diferencias entre tres computadoras con diferentes especificaciones para el caso de $n = 1, 2, ..., 35$.

2. Algoritmo

Para obtener más información sobre el problema de las torres de Hanoi ver [2]. El siguiente código en Python implementa la solución por métodos recursivos de este problema.

Listing 1: Solución Recursiva de las Torres de Hanoi

```
def hanoi(n, from_rod, to_rod, aux_rod):
        if n == 1:
                # print "Move disk 1 from rod", from_rod, "to rod", to_rod
                return
        hanoi(n-1, from_rod, aux_rod, to_rod)
        # print "Move disk", n, "from rod",from_rod, "to rod", to_rod
        hanoi(n-1, aux_rod, to_rod, from_rod)
```

Sin embargo, para poder medir los pasos y el tiempo de ejecución, se utilizó el siguiente código que modifica el anterior:

Listing 2: Conteo de Pasos y Tiempos

```
import pickle
from datetime import datetime
count = 0

def hanoi(n, from_rod, to_rod, aux_rod):
        global count
        count = count + 1
        if n == 1:
                # print "Move disk 1 from rod", from_rod, "to rod", to_rod
                return
        hanoi(n-1, from_rod, aux_rod, to_rod)
        # print "Move disk", n, "from rod",from_rod, "to rod", to_rod
```

```
        hanoi(n-1, aux_rod, to_rod, from_rod)


trials = 35
steps = []
times = []
for i in range(1, trials + 1):
        count = 0
        startTime = datetime.now()
        hanoi(i, 'A', 'B', 'C')
        times.append((datetime.now() - startTime).total_seconds())
        steps.append(count)

with open("times.txt", "wb") as times_file:
        pickle.dump(times,times_file)

with open("steps.txt", "wb") as steps_file:
        pickle.dump(steps,steps_file)
```

Como es sabido, la complejidad de esta solución es $O(2^n)$. Esto se puede ver en el conteo de pasos para los diferentes niveles de n junto con la fráfica de la ecuación $y = 2^x$.

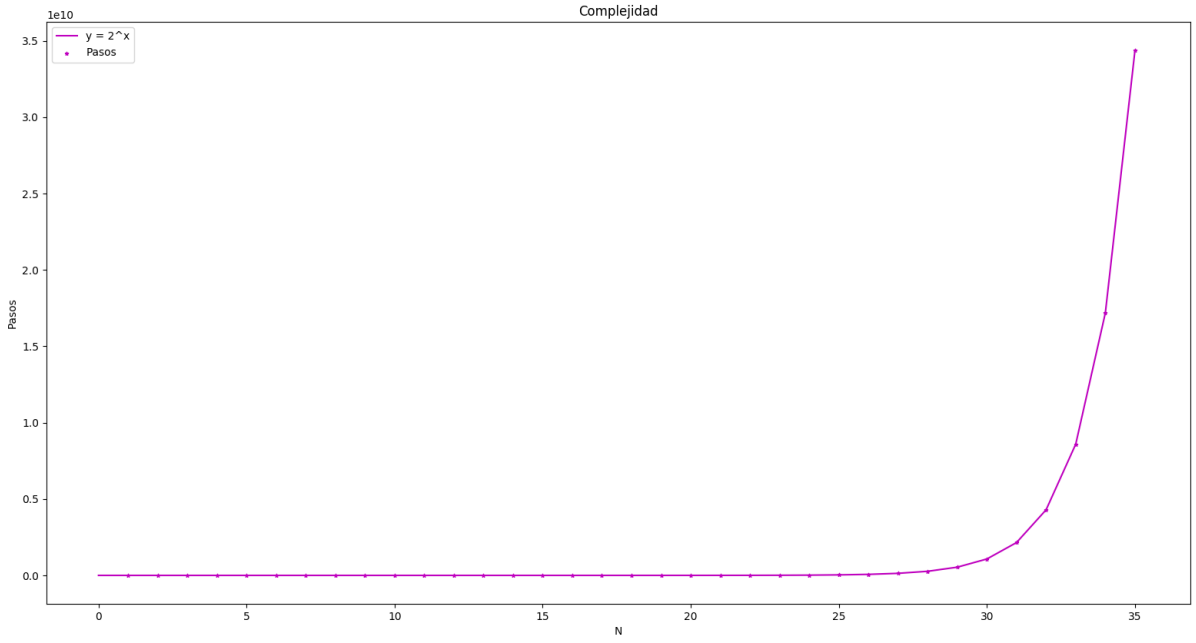

Figura 1: Complejidad del Algoritmo

3. Niveles de Cómputo

Para realizar el experimento se utilizaron, primero, dos *droplets* de DigitalOcean con las siguientes especificaciones.

```
model name      : Intel(R) Xeon(R) CPU E5-2697A v4 @ 2.60GHz
```

Figura 2: CPU 2.60 GHz

MEMORY	DEDICATED vCPUs	SSD DISK
4 GB	**2 vCPUs**	25 GB

Figura 3: Droplet 1

Puede notarse que aunque habían disponibles 2 CPUs sólo se utilizó uno en su máxima capacidad y por eso el poder de cómputo total fue de sólo 50 %.

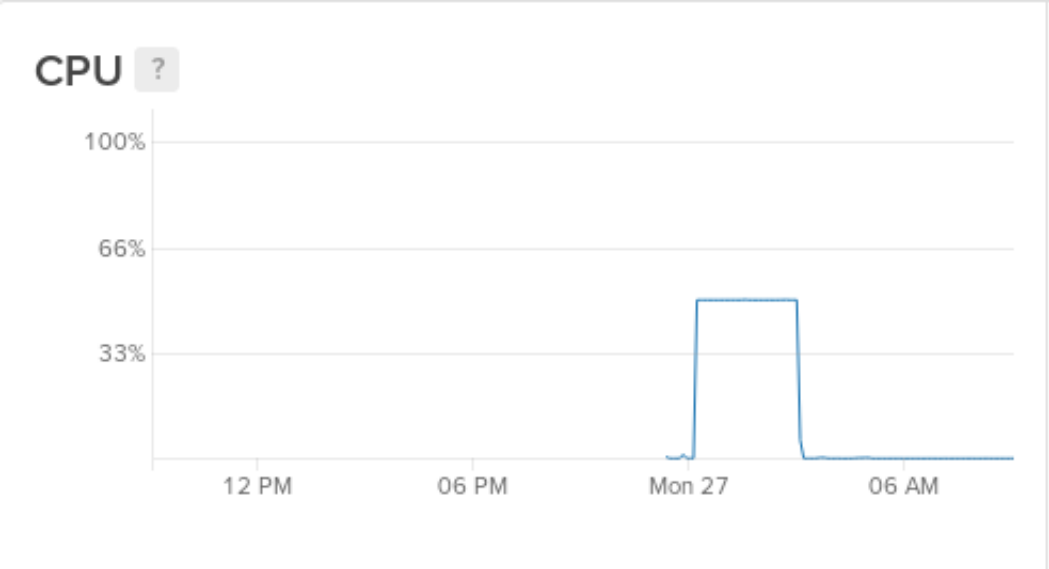

Figura 4: Desempeño de Droplet 1

```
model name      : Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
```

Figura 5: CPU 2.20 GHz

4 GB	2 vCPUs	80 GB

Figura 6: Droplet 2

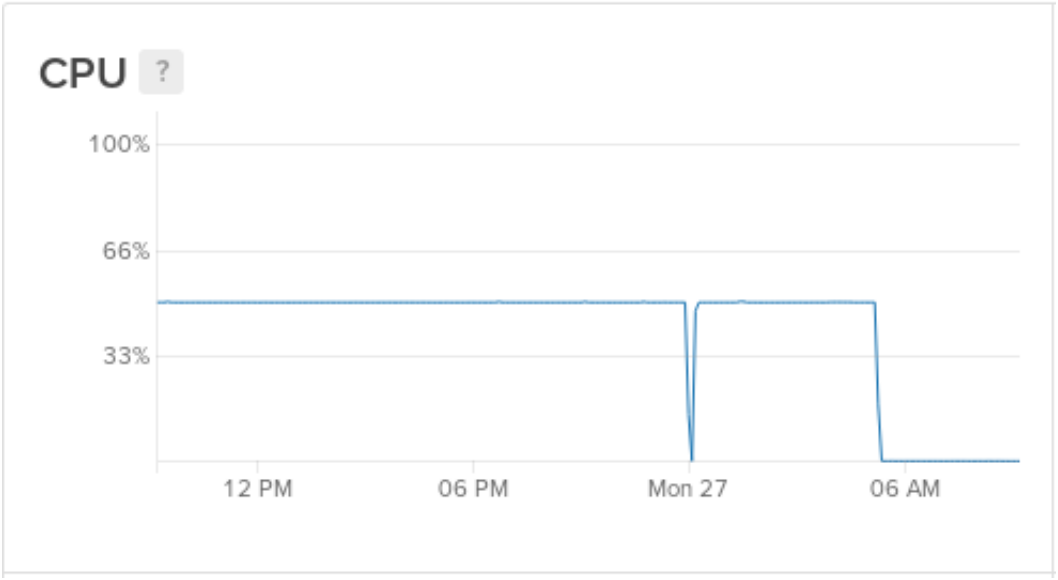

Figura 7: Desempeño de Droplet 2

4. Resultados

El tiempo de corrida del algoritmo en los diferentes droplets puede verse en la siguiente gráfica.

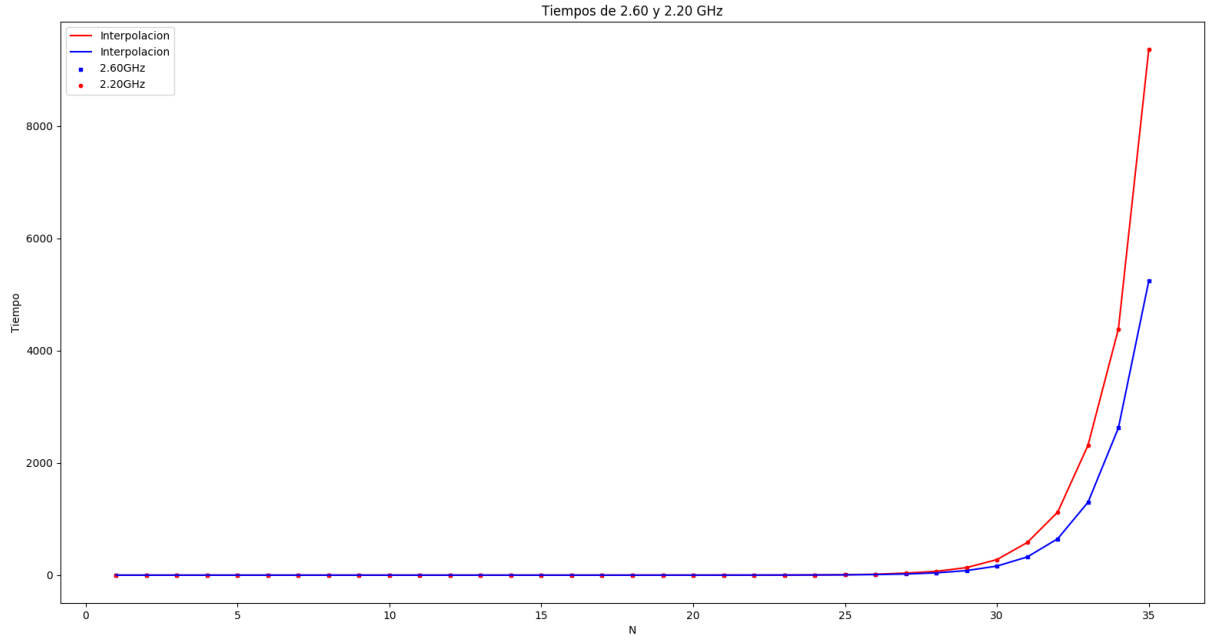

Figura 8: Diferencia de Tiempos

5. Discusión

La diferencia en los tiempos de ejecución entre los dos droplets es esperada. Si computamos para los 35 puntos el cociente entre el tiempo tomado por el CPU de 2.20 GHz y el tiempo tomado por el CPU de 2.60 GHz y tomamos el promedio, obtenemos aproximadamente 1.63. Si tomamos el cociente entre los GHz 2.60/2.20 obtenemos aproximadamente 1.18. La diferencia puede ser debido a que, de acuerdo con DigitalOcean, los CPUs de 2.60 GHz están optimizados: *CPU Optimized Droplets are best used for CPU intensive projects that require predictable performance, such as batch processing large data sets, builds of large processes, video encoding, and other projects that rely on CPU more than RAM or I/O.* Además, esta medida es relativamente buena ya que aunque visualmente parece que la diferencia entre tiempos es mayor conforme n crece, en la realidad la desviación estándard de los cocientes de tiempos es 0.47, que representa tan sólo un .29 del promedio.

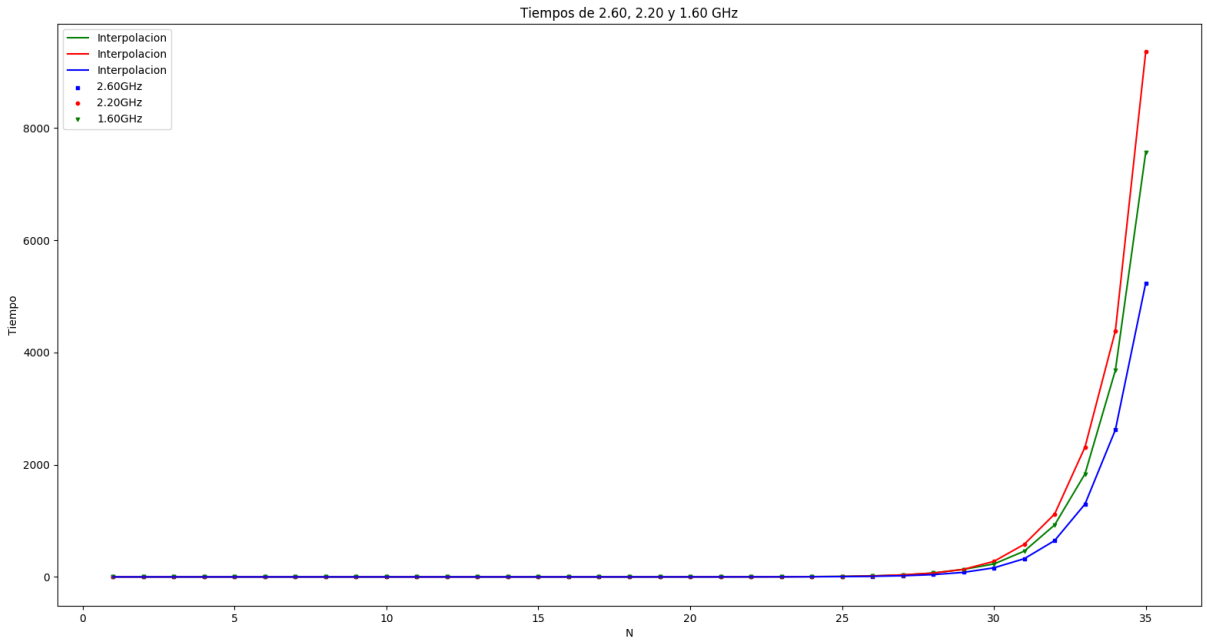

Figura 9: Diferencia de Tiempos

Para probar la robustez de esos resultados se corrió el algoritmo una tercera vez con un procesador de 1.60 GHz. Sin embargo, esta vez se encontró que aunque los tiempos fueron mayores que los del droplet de 2.60 GHz fueron a su vez más bajos que los del droplet de 2.20 GHz. Podemos afirmar, sin embargo, que esto se debe a que los CPUs tienen diferentes arquitecturas. En [3] se realiza un benchmark para analizar estas diferencias:

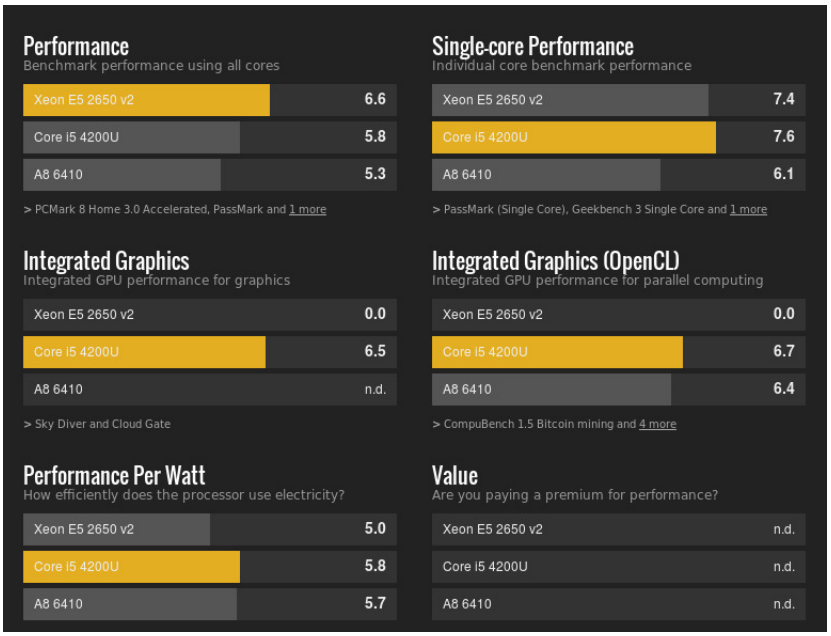

Figura 10: Diferencia de Tiempos

6. Conclusión

Computadoras con diferentes niveles de poder computacional pueden exhibir diferente desempeño. Sin embargo, el nivel de GHz sólo es una buena referencia si se utiliza la misma arquitectura

Referencias

[1] Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford, *Introduction to Algorithms, Third Edition*, The MIT Press, 2009

[2] `https://en.wikipedia.org/wiki/Tower_of_Hanoi`

[3] `http://cpuboss.com/cpus/Intel-Xeon-E5-2650-v2-vs-Intel-Core-i5-4200U`