Trabajo Práctico Programación Concurrente

Participantes:

Duglas Español Coloma, Legajo Nº 42237, duglas.espanol@gmail.com

Jonathan Gutierrez, Legajo Nº 39817, jonathan.a.gutierrez9@gmail.com

Introducción:

Este trabajo se basa en el algoritmo de “Proof of work” de la red de Bitcoin utilizando Java. Se diseñó un programa con el objetivo de conseguir un valor de nonce tal que los primeros n bits del resultado de aplicar la función hash Sha-256 a la secuencia de bytes sean 0.

Respecto al diseño del código las clases involucradas en el trabajo son:

- Buffer = En este caso lo más importante son las operaciones synchronized de read y write a la cual en la primera es usada por los workers y la última por el main a la hora de repartir las unidades de trabajo

- PowWorker = Este se encarga de pedirle al buffer una unidad de trabajo para poder trabajar y recorrer los números con un while, en cada iteración se comprueba si al resultado de aplicarle al (prefijo + índice) el sha 256 nos da sus primeros bytes en 0 basándonos en la dificultad. Los workers tienen un booleano (encontrado) que es usado para mantener el while y marca cuando el worker encontró el nonce.
  Algo que decidimos fue que el worker tenga en el run el inicio del contador y tanto si encontró el nonce correcto como si no lo encontró corta el timer y lo devuelve.

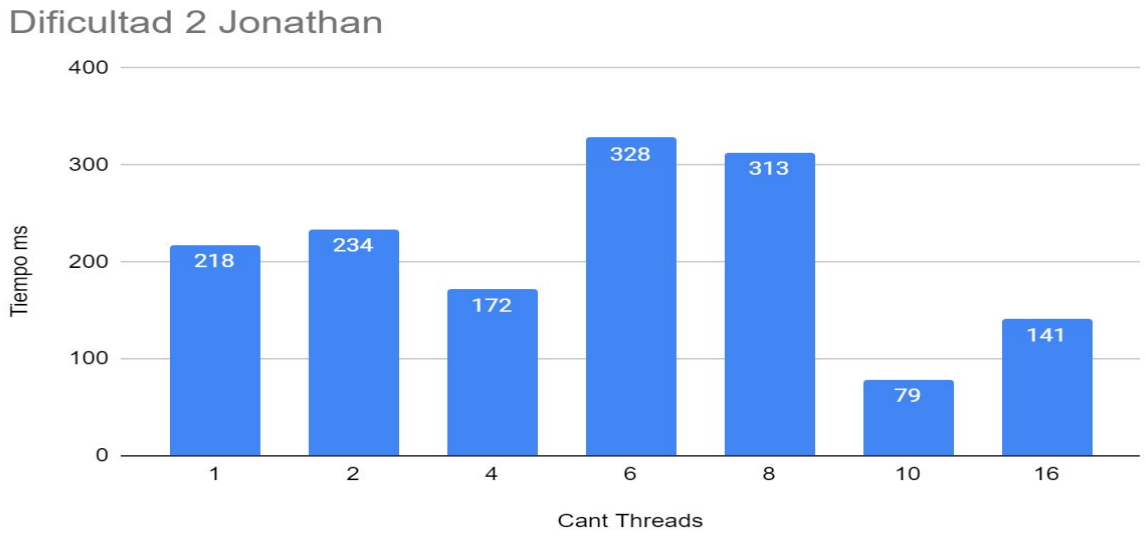
- ThreadPool = En esta clase lo que hacemos es crear y arrancar los workers que se van a encargar de realizar la búsqueda del nonce.Y cuando uno de estos PowWorkers encuentren el nonce correcto van a comunicarselo al threadpool, y este como conoce a todos los threads, les envía a todos un mensaje para hacer que detengan la búsqueda del nonce.

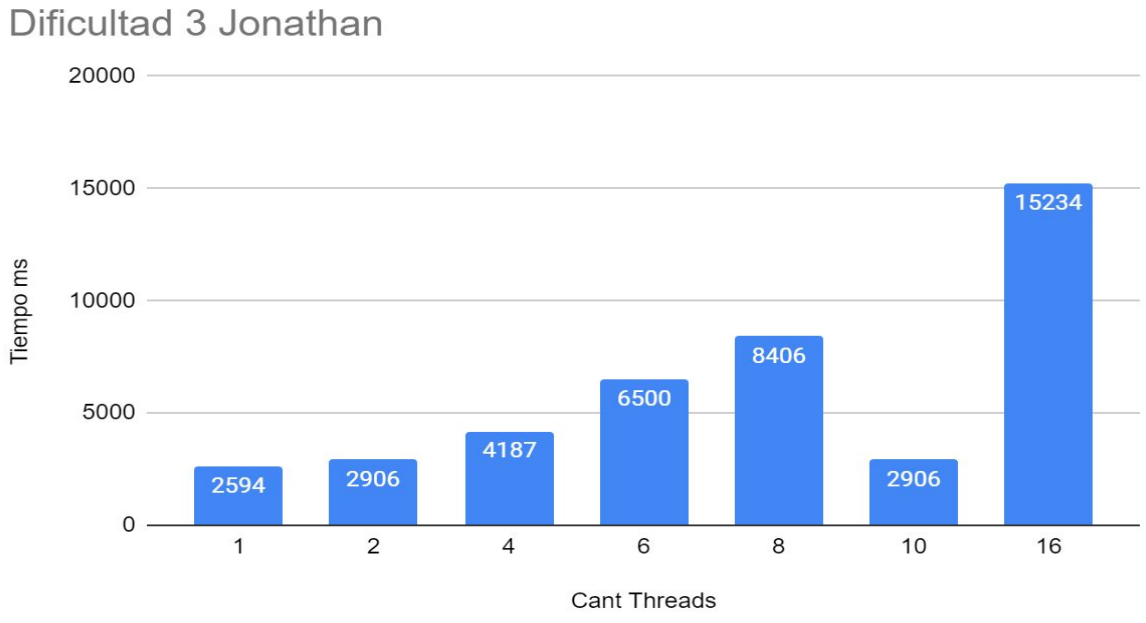
- UnidadDeTrabajo = Simplemente tiene el mínimo, el máximo del rango que le será provisto por la distribución que reciba del main, y también tiene el texto prefijo para concatenar/juntar con el nonce

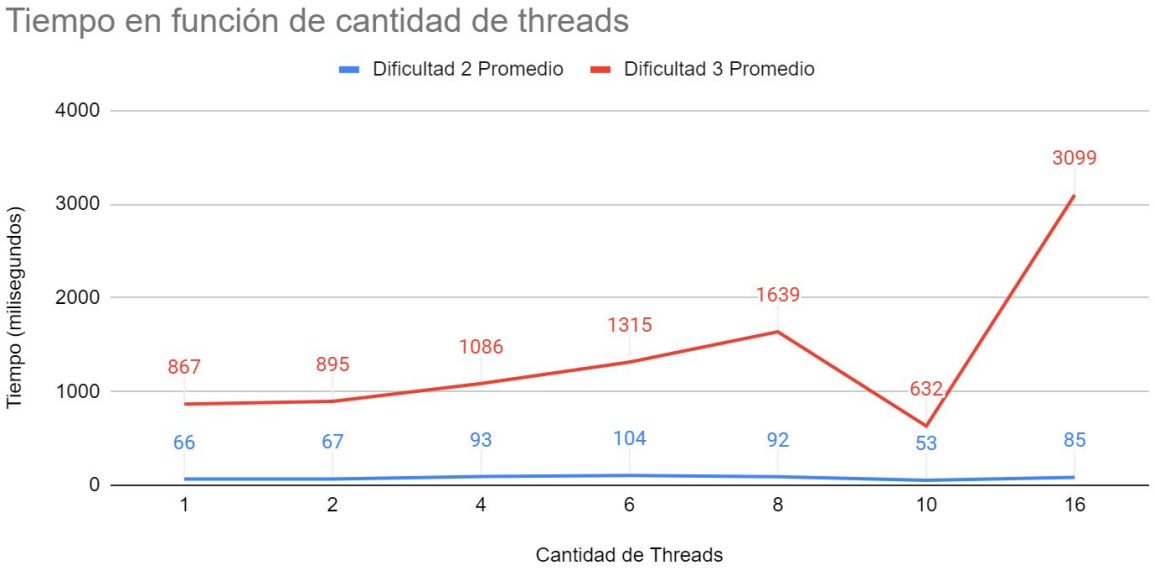
- Main = Aquí pedimos los inputs del usuario para obtener la cantidad de threads para trabajar, la dificultad, y un prefijo.Se instancia el buffer y asi como el Thread Pool.Tambien aca hacemos el cálculo de cada unidad de trabajo basado en la cantidad de threads para que sea equitativo

Evaluación:

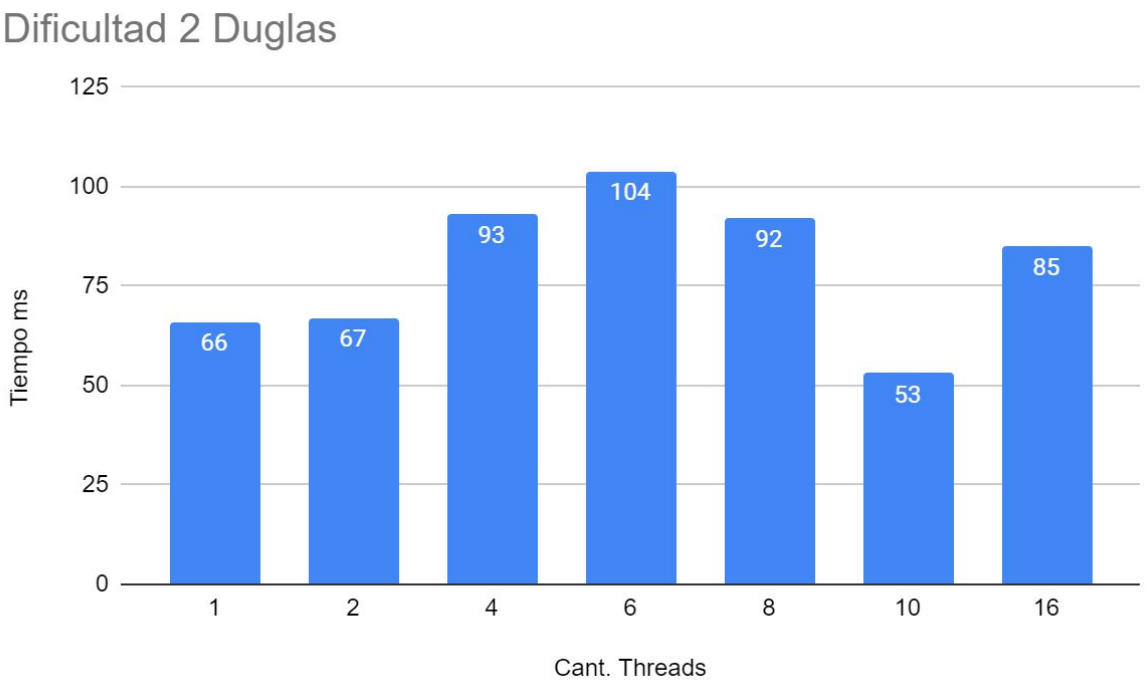
Para la evaluación del funcionamiento del programa se realizo búsqueda de nonce en dificultades 2 y 3 para 1, 2, 4, 6, 8, 10 y 16 threads.Tratando de identificar la cantidad de threads óptima. Se utilizó la siguiente computadora: Jonathan Notebook =
Intel Core I3-4005U @1.70ghz; 12 gb ram ; Windows 10 Pro Version 2004.
Se Obtuvieron lo siguientes resultados

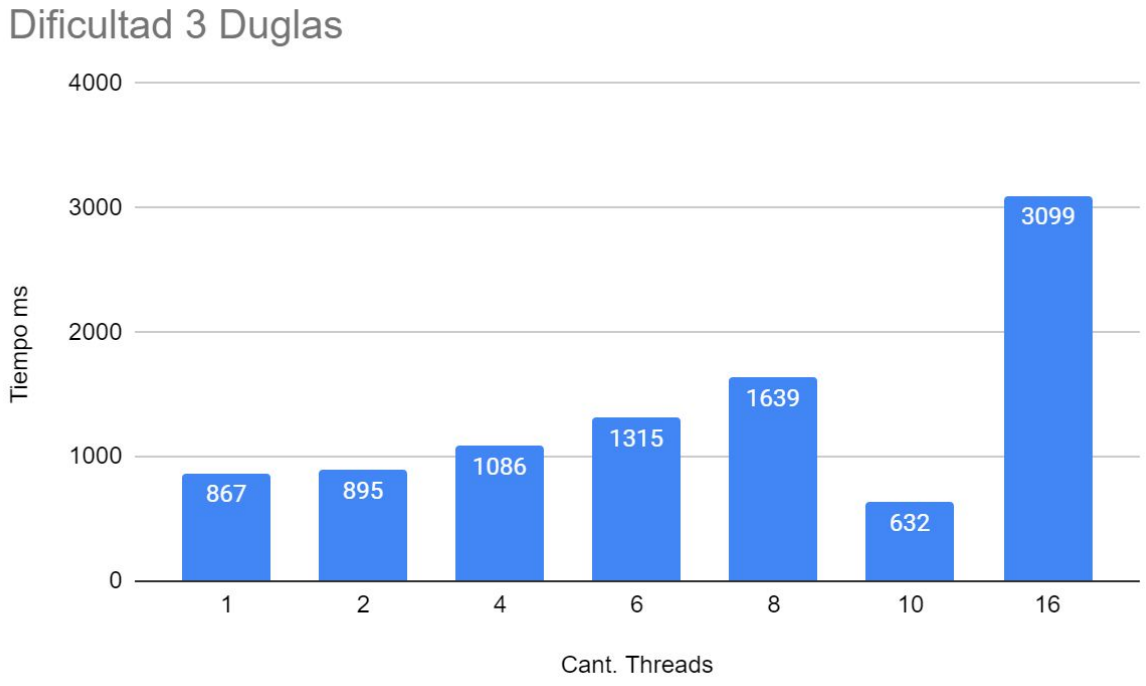


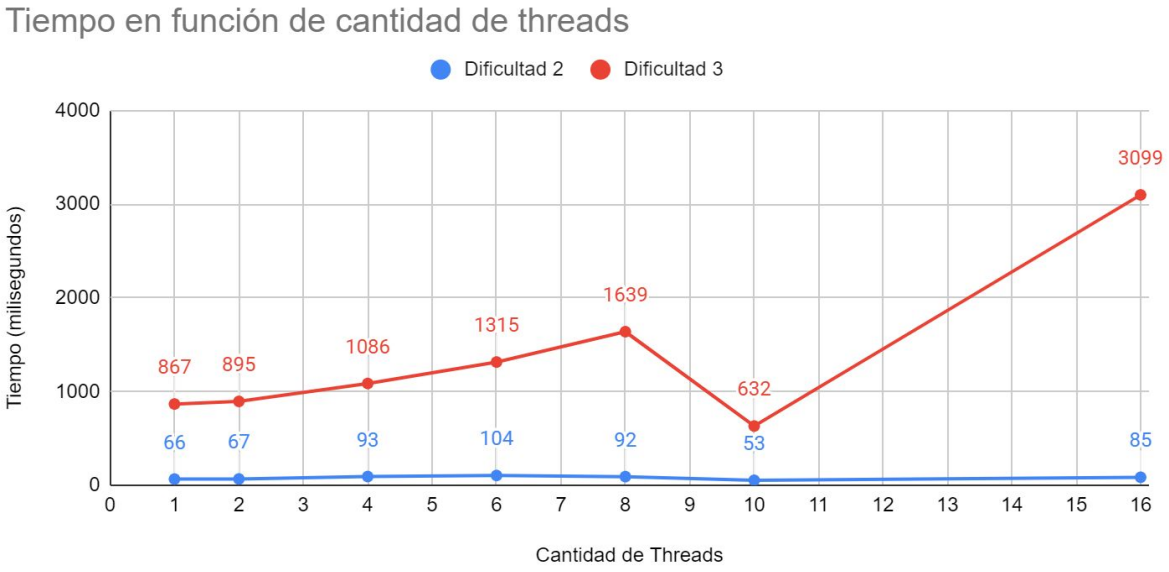



Tiempo en función de cantidad de threads
Dificultad 2 Promedio
Dificultad 3 Promedio
4000
3000
2000
1000
0
Tiempo (milisegundos)
867
895
1086
1315
1639
632
3099
66
67
93
104
92
53
85
1
2
4
6
8
10
16
Cantidad de Threads

Utilizando la computadora de Duglas (Intel(R) Core(™) i5-8250U CPU @ 1.60GHz 1.80 GHz; 8,00 GB RAM; Windows 10 Home Single Language version 2004) Se obtuvo:






Tiempo en función de cantidad de threads
Dificultad 2
Dificultad 3
Tiempo (milisegundos)
4000
3000
2000
1000
0
867
895
1086
1315
1639
632
3099
66
67
93
104
92
53
85
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Cantidad de Threads

Análisis:

En nuestro caso se dio que 10 threads es la cantidad óptima sin importar cual de las de las computadoras se utilizara. Al ver ambas curvas se ve que siguen la misma forma, al aumentar la cantidad de threads aumenta el tiempo de ejecución exceptuando la cantidad de threads optima (10), en donde se da el mínimo tiempo de ejecución.

Al buscar el golden nonce, en la computadora de Duglas, con la cantidad óptima de threads y el string “”, se dio que después de aproximadamente 11 minutos se encontró el golden nonce, que fue el 1698877448. Mientras que en la de Jonathan se tardó 28 m.
En cambio en el caso del prefijo “hola” en la notebook de Jonathan, se encontró el golden nonce tardandose 23 m mientras que en la de Duglas se tardó aproximadamente 8 minutos. En este caso el golden nonce fue 3709964314.