Proyecto Final

Computación en Internet I

Depto. Ciencias Físicas y Exactas / Facultad de Ingeniería, Diseño y Ciencias Aplicadas

Profesor
Nicolás Javier Salazar Echeverry

Daniela Castaño Moreno
Simón García Zuluaga
Juan José Ramos Henao

Universidad ICESI
Cali, Valle del Cauca
2025

Descripción del problema

Un número perfecto es aquel que es igual a la suma de sus divisores propios positivos, excluyéndose a sí mismo. Por ejemplo, 28 es perfecto porque 1 + 2 + 4 + 7 + 14 = 28. Este proyecto busca encontrar todos los números perfectos en un rango específico, de forma eficiente y paralela.
Debido a que el problema es altamente costoso computacionalmente (su complejidad es $O(n\sqrt{n})$), se utilizará una arquitectura distribuida donde:

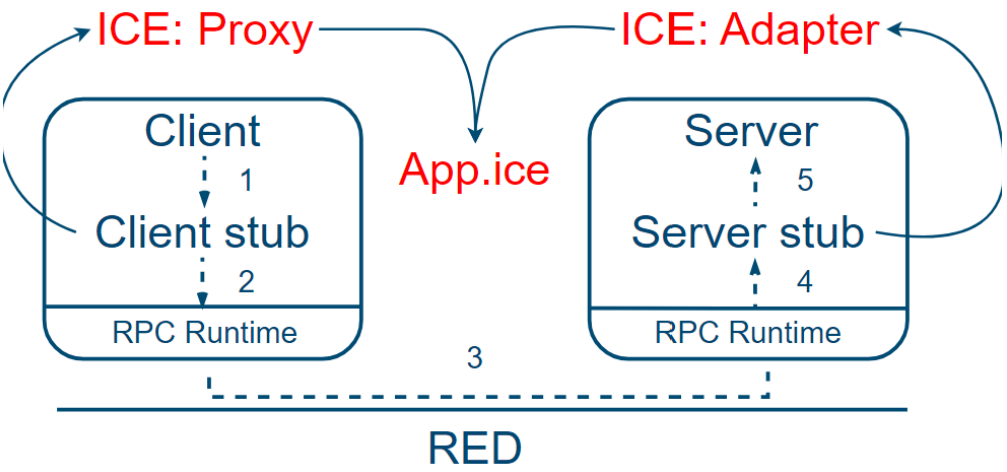
- Un cliente solicita encontrar todos los números perfectos en un rango determinado.
- Un maestro divide el rango en subrangos para ser procesados.

Múltiples trabajadores procesan cada subrango y devuelven los números perfectos encontrados al maestro.

Teoría relevante

RPC - Remote Procedure Call

La idea detrás de RPC es hacer que una llamada a un procedimiento remoto se parezca tanto como sea posible a una llamada local, intentando abstraer las operaciones de red a un alto nivel. En su forma más simple, para llamar a un procedimiento remoto, el programa cliente debe estar enlazado con un pequeño procedimiento de biblioteca, llamado stub del cliente, en Ice este es conocido como un Proxy, que representa el procedimiento del servidor en el espacio de direcciones del cliente. Del mismo modo, el servidor está vinculado a un procedimiento llamado stub del servidor, en Ice es conocido como el Adapter. Estos procedimientos ocultan el hecho de que la llamada al procedimiento del cliente al servidor no es local. El siguiente diagrama muestra un resumen de los componentes de un RPC, y cómo se traducen a Ice:



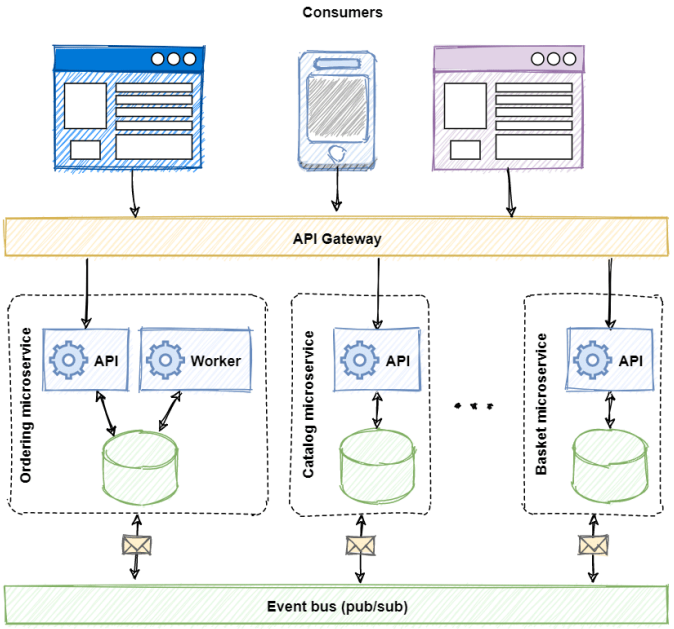
Algoritmo utilizado

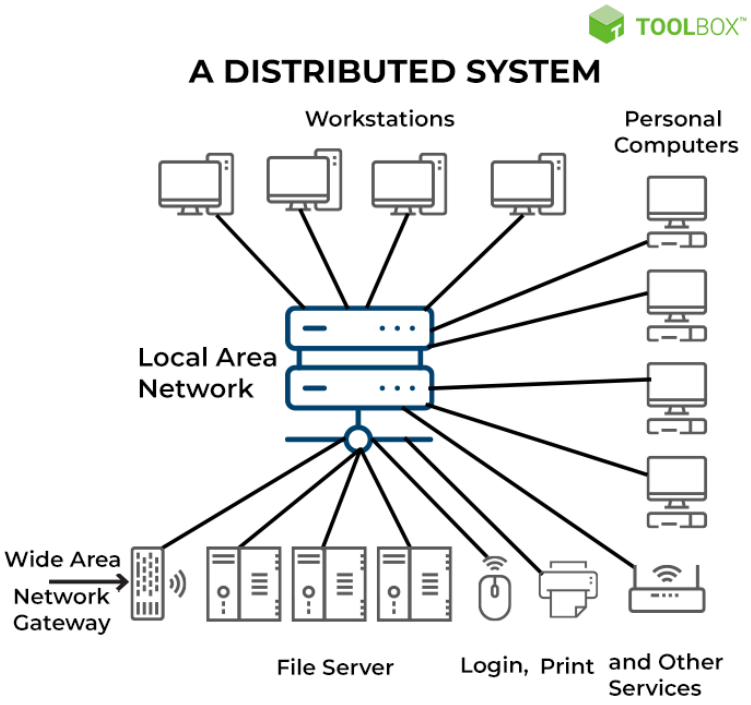
El algoritmo que utilizamos está basado en distribución de responsabilidades con el diseño cliente-maestro con ICE, donde tenemos un nodo que se encarga de la distribución de rangos (algoritmo de distributeNodes) y los otros nodos se encargan de aplicar el algoritmo que detecta qué números en ese rango son números perfectos (algoritmo calculatePerfectNumerbers).

Con Zero Ice, Gradle y Java logramos realizar un proyecto que encuentra números perfectos en un rango dado por un cliente, este se le pasa al master quien reparte los rangos equitativamente a los workers y estos arrojan los números perfectos en su rango asignado.

Arquitectura general del sistema distribuido

Un sistema distribuido es un conjunto de programas informáticos que utilizan recursos computacionales en varios nodos de cálculo distintos para lograr un objetivo compartido común. Este tipo de sistemas, también denominados "computación distribuida" o "bases de datos distribuidas", usan nodos distintos para comunicarse y sincronizarse a través de una red común. Estos nodos suelen representar dispositivos de hardware físicos diferentes, pero también pueden representar procesos de software diferentes u otros sistemas encapsulados recursivos. La finalidad de los sistemas distribuidos es eliminar los cuellos de botella o los puntos de error centrales de un sistema**.**



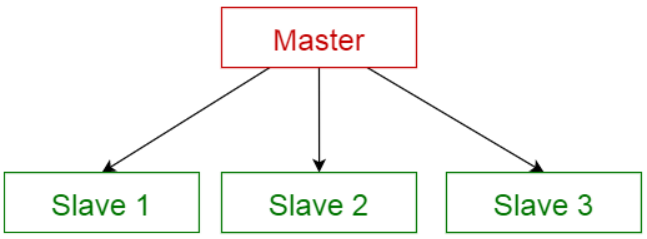


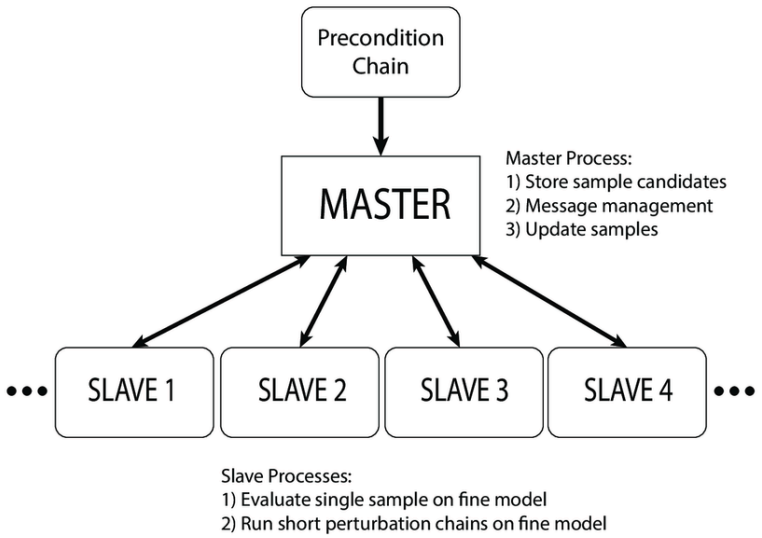
Diseño Cliente-Maestro con ICE

- *Cliente:* Se conecta con el Máster (Servidor), envía el rango para analizar los números perfectos, y la cantidad de nodos (cantidad de workers).
- *Master:* Recibe los datos del cliente, divide el rango en base a los workers requeridos y envía los nuevos rangos a los workers. Después recibe los resultados de los workers y los devuelve al cliente.
- *Worker:* Recibe un rango del servidor, y encuentra todos los números perfectos en este rango, y devuelve el resultado al servidor.

El cliente es "el Main" de la aplicación (todos son subprocesos, cada uno tiene su método "main", pero el cliente es quien empieza todo).

(Falta imagen / foto de mi galería)





Mecanismo de distribución del rango y coordinación

El maestro separa los rangos de manera equitativa y controla a los esclavos y se asegura de que el bus de datos reciba y procese la información correctamente, con la siguiente estrategia:

Digamos que tenemos un rango desde 0 a 22 (son 23 números en total), y tenemos 3 workers.
Entonces lo que se hace es 23/3 (esto da 7, con un residuo de 2).
Primero, el maestro define un rango con base en la división.

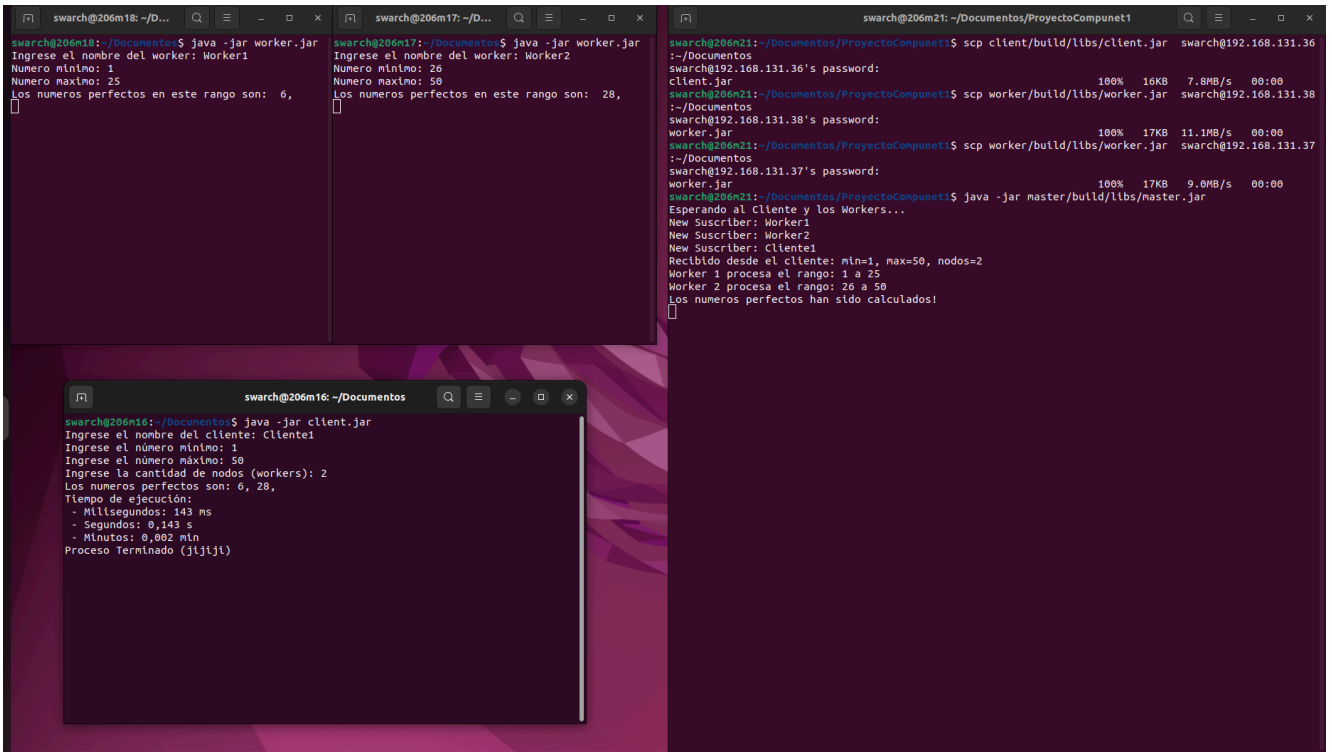
Worker 1: 0-6 (7 números)
Worker 2: 7-13 (7 números)
Worker 3: 14-20 (7 números)

Esto en total nos da 21 números, pero son 23.

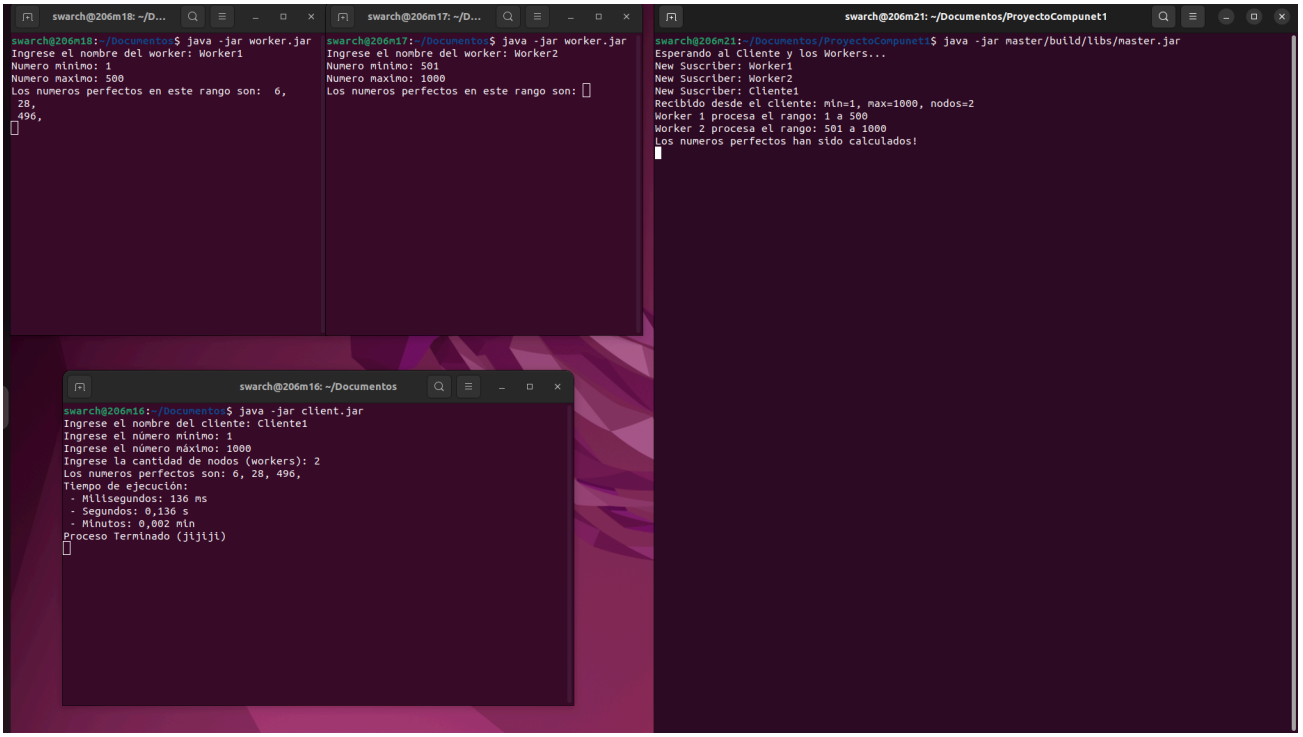
Por lo tanto, después se suman los residuos, y al final tenemos:
Worker 1: 0-7 (8 números)
Worker 2: 8-15 (8 números)
Worker 3: 16-22 (7 números)

Resultados experimentales y análisis de rendimiento

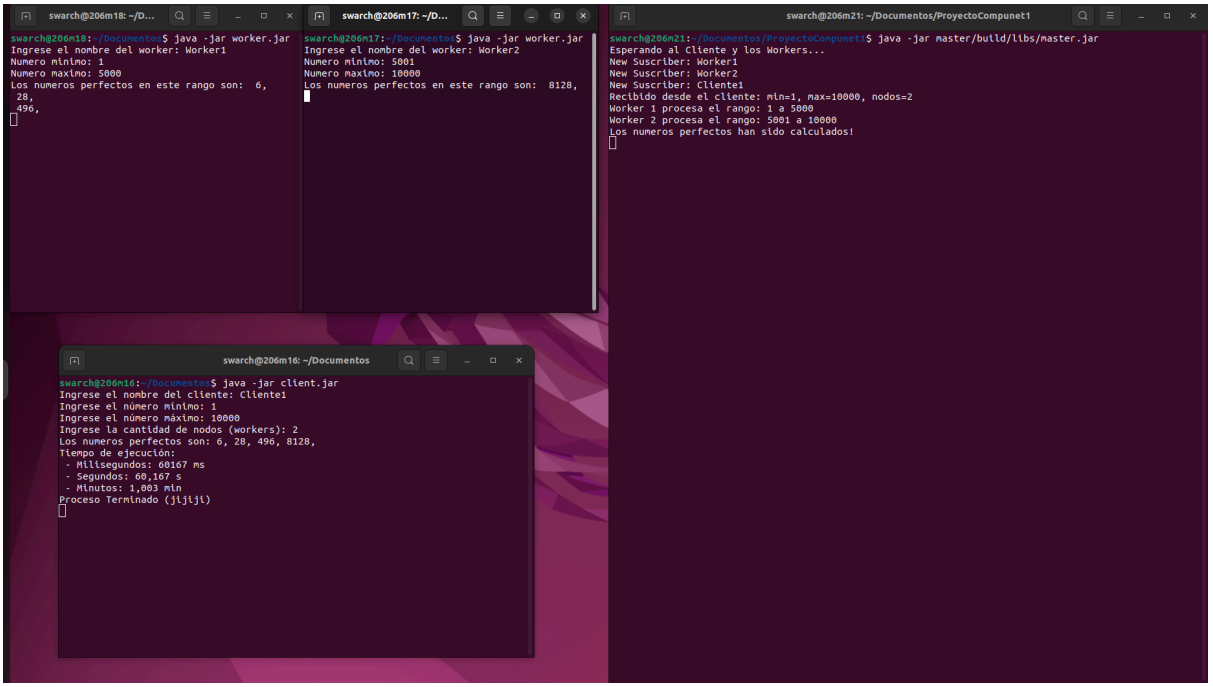
[1,50] - 2 workers - Sincrónico



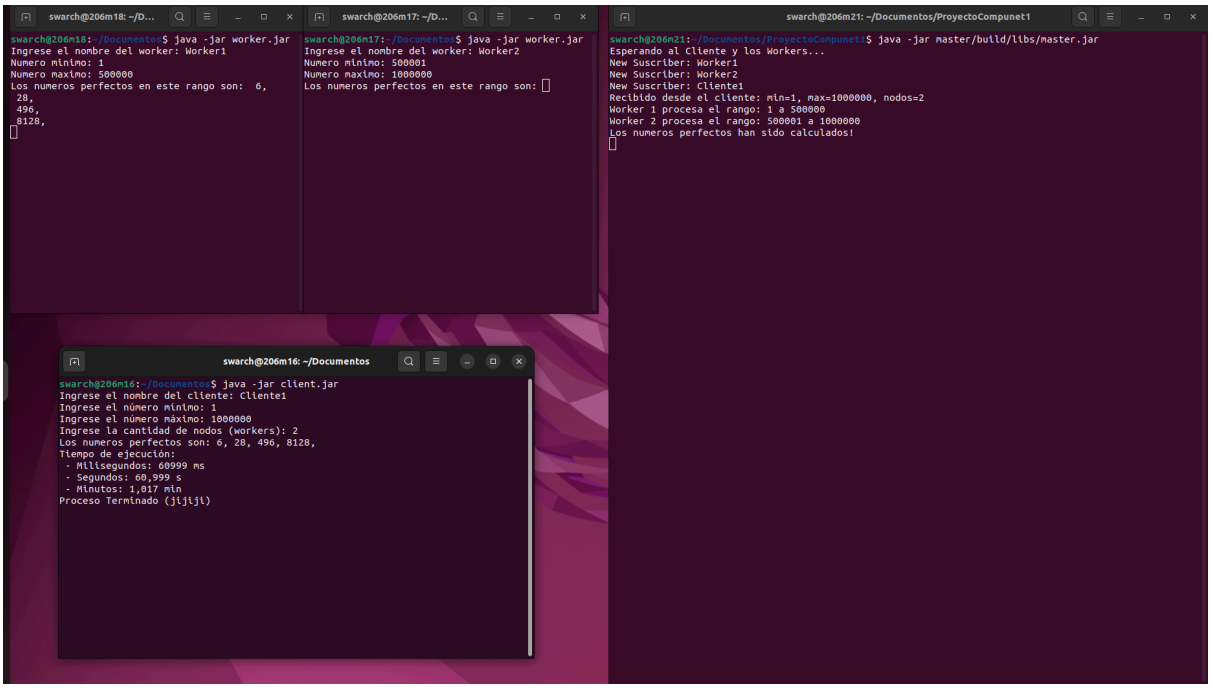
[1,1000] - 2 workers - Sincrónico



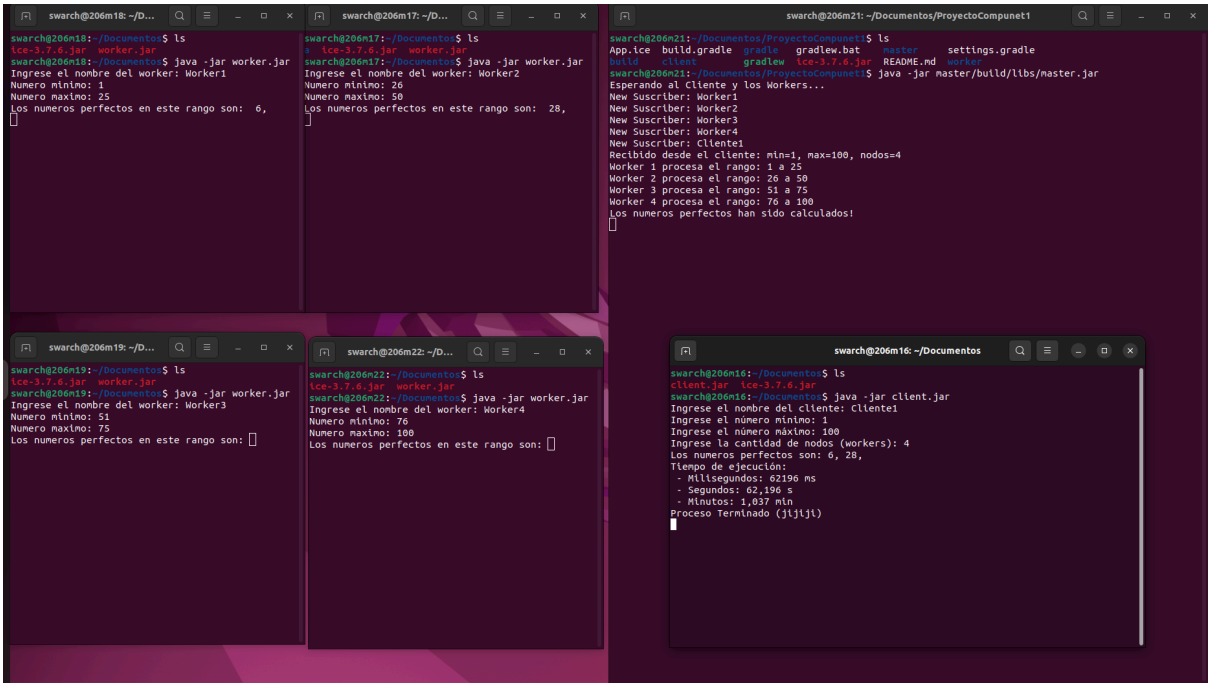
[1,10000] - 2 workers - Sincrónico



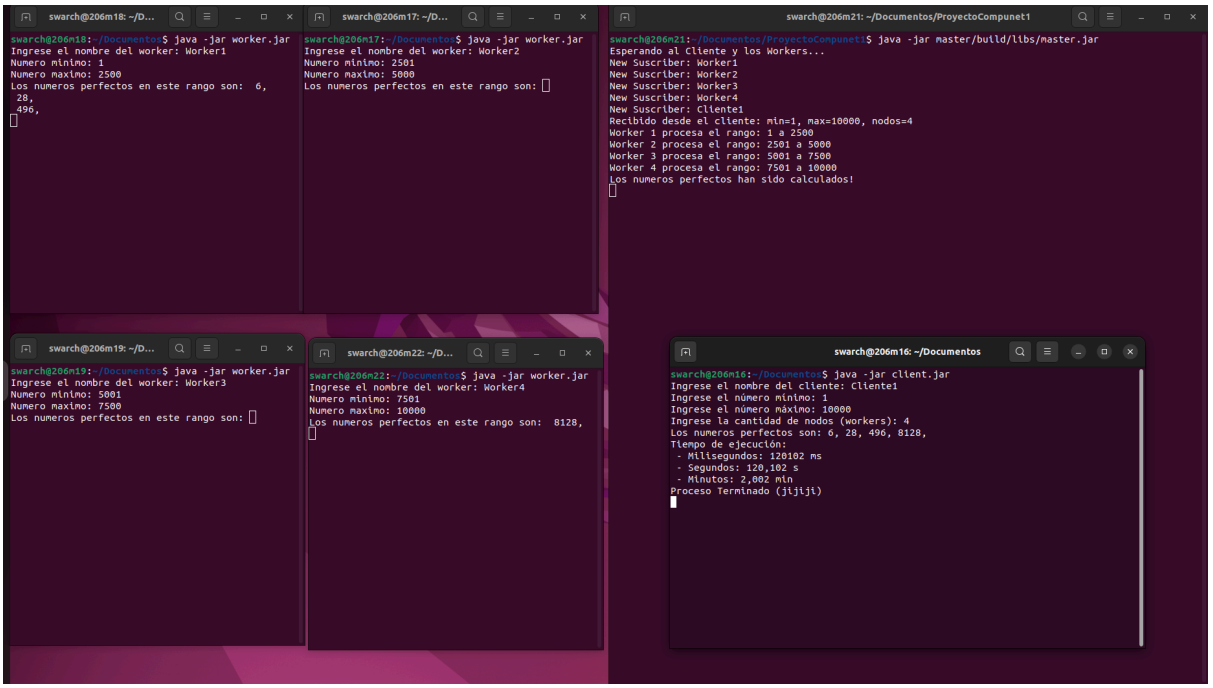
[1,1000000] - 2 workers - Sincrónico



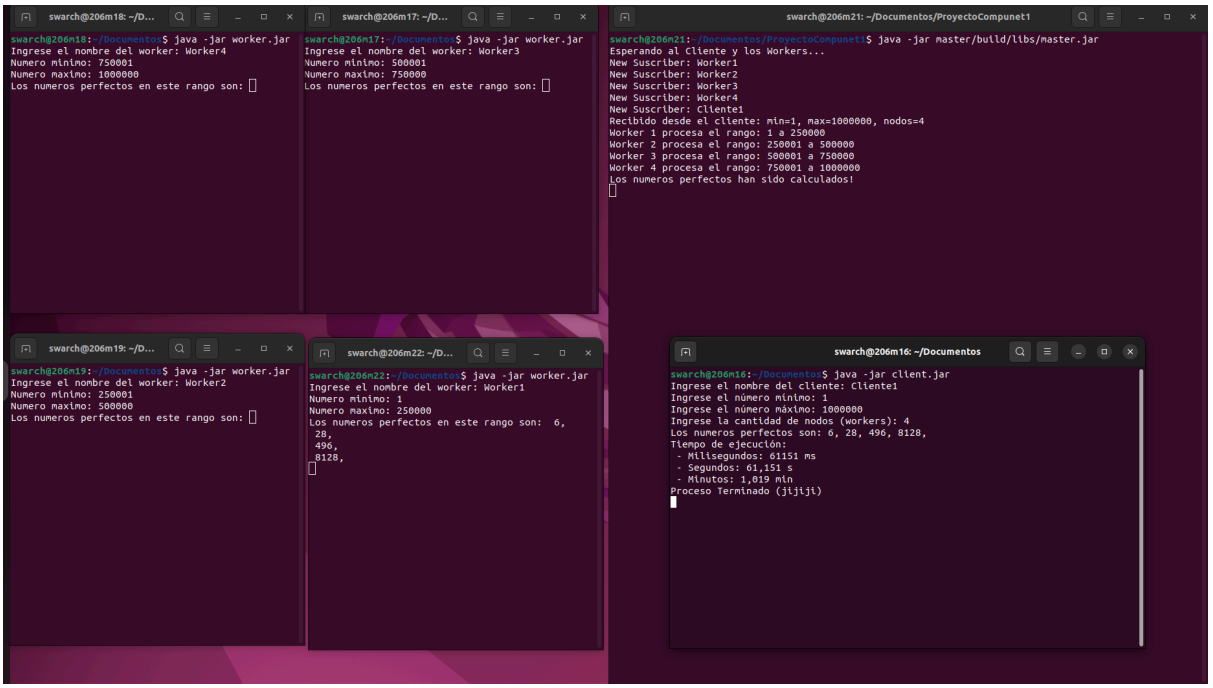
[1,100] - 4 workers - Sincrónico



[1,10000] - 4 workers - Sincrónico



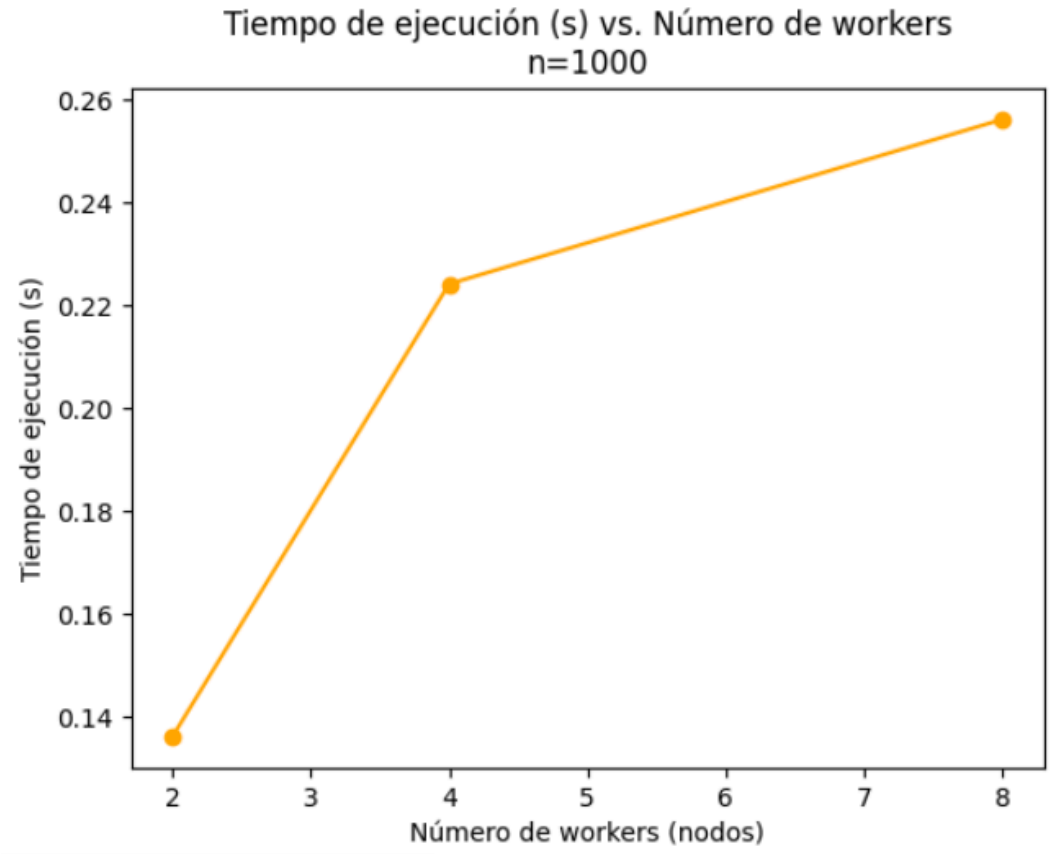
[1,1000000] - 4 workers - Sincrónico

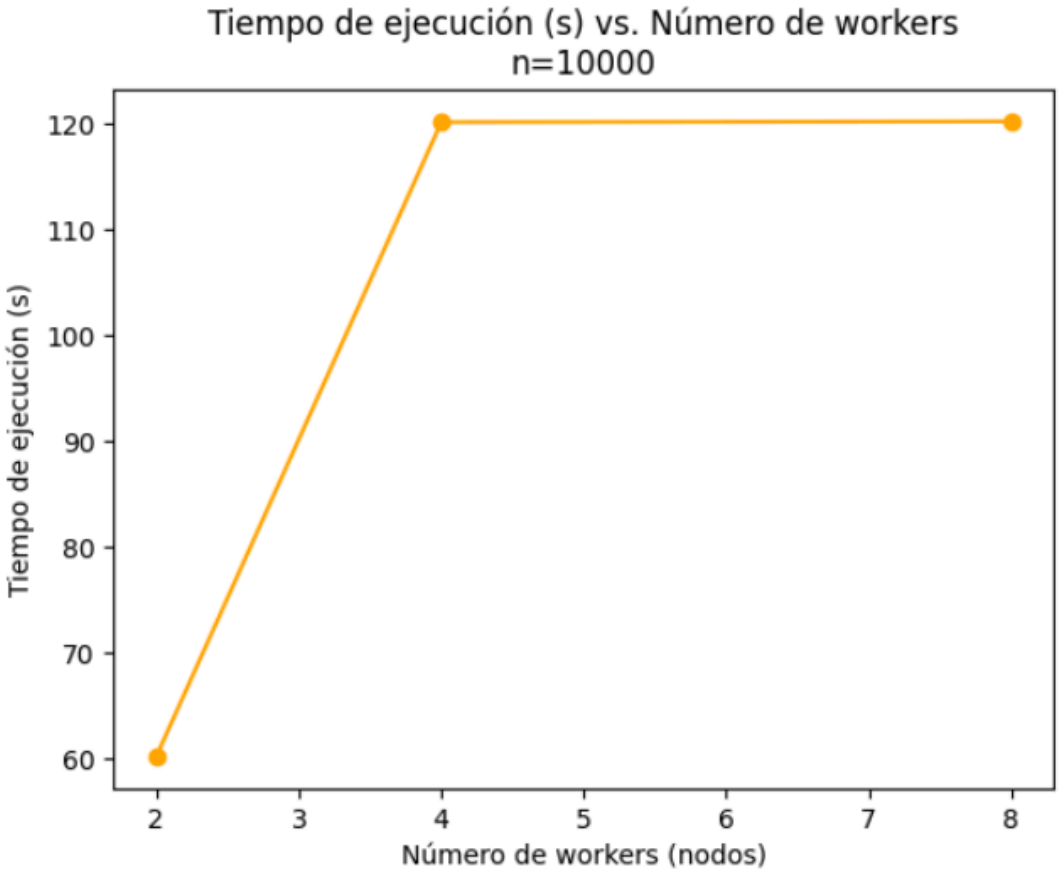


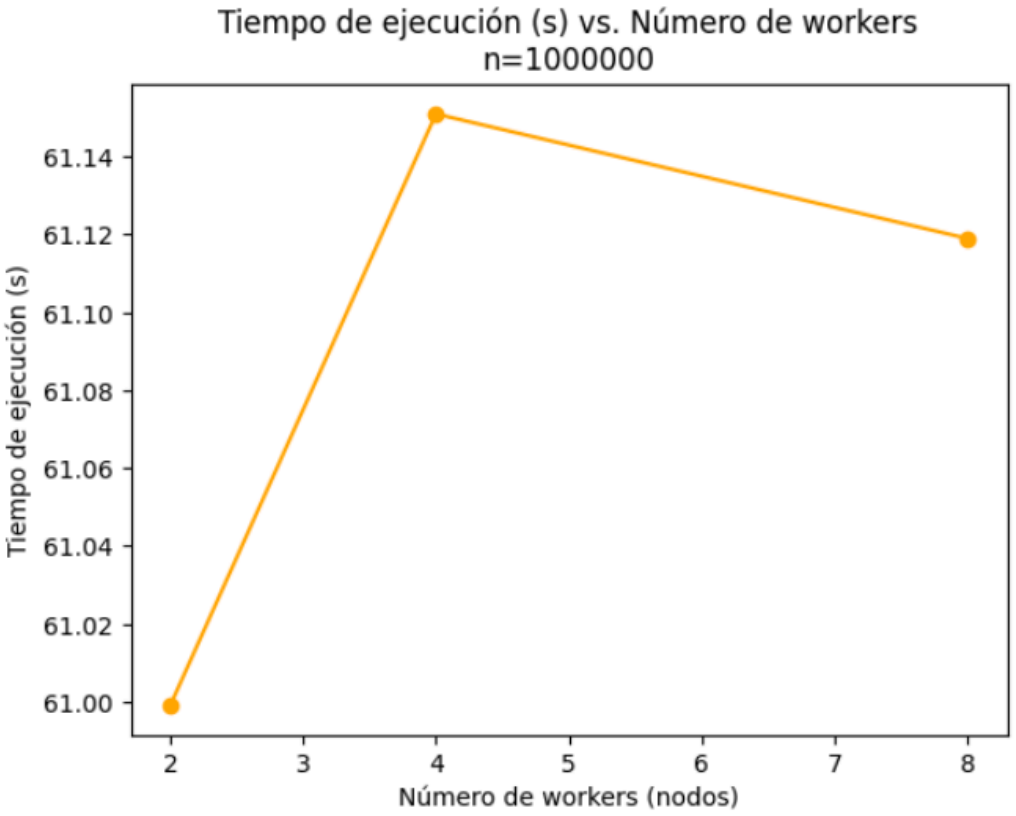
Medición y reporte del tiempo total de procesamiento y la eficiencia

Número de workers	Inicio del rango	Fin del rango	Tiempo de ejecución en milisegundos (ms)	Tiempo de ejecución en segundos (s)
2	1	1000	136 ms	0,136 s
2	1	10000	60167 ms	60,167 s
2	1	1000000	60999 ms	60,999 s
4	1	1000	224 ms	0,224 s
4	1	10000	120102 ms	120,102 s
4	1	1000000	61151 ms	61,151 s
8	1	1000	256 ms	0,256 s
8	1	10000	120173 ms	120,173 s
8	1	1000000	6119 ms	61,119 s

Gráficas número de Tiempo vs. Número de workers







Para evaluar el rendimiento utilizaremos la siguiente fórmula:

$$Eficiencia(Rendimiento) = \frac{tiempoEjecución}{\#workers}$$

Con esto mediremos la eficiencia o rendimiento de la ejecución dependiendo la cantidad de workers, para validar si la distribución de trabajo es viable. Se mide el tiempo "promedio".

La eficiencia es la capacidad de un objeto, un proceso o una persona de alcanzar sus objetivos de la mejor manera posible.

$n = 1000$
$workers = 2$

$$\frac{0.136}{2} = 0.068$$

$n = 1000$
$workers = 4$

$$\frac{0.224}{4} = 0.056$$

$n = 1000$
$workers = 8$

$$\frac{0.256}{8} = 0.032$$

Para n=1000 la mejor eficiencia o distribución de trabajo para los nodos o workers fue 8 workers, ya que cada nodo tardó menos tiempo en realizar su trabajo.

$n = 10000$
$workers = 2$

$$\frac{60.167}{2} = 30.084$$

$n = 10000$
$workers = 4$

$$\frac{120.102}{4} = 30.026$$

$n = 10000$
$workers = 8$

$$\frac{120.173}{8} = 15.022$$

Para n=10000 la mejor eficiencia o distribución de trabajo para los nodos o workers fue 8 workers, ya que cada nodo tardó menos tiempo en realizar su trabajo.

$n = 1000000$
$workers = 2$

$$\frac{60.999}{2} = 30.500$$

$n = 1000000$
$workers = 4$

$$\frac{61.151}{4} = 15.288$$

$n = 1000000$
$workers = 8$

$$\frac{61.119}{8} = 7.640$$

Para n=10000 la mejor eficiencia o distribución de trabajo para los nodos o workers fue 8 workers, ya que cada nodo tardó menos tiempo en realizar su trabajo.

Conclusiones y posibles mejoras

Con la realización de este proyecto logramos entender lo básico, la introducción sobre qué es Ice, esto junto con Gradle y el lenguaje de programación Java. Para nosotros fue un nuevo aprendizaje con el cual debemos seguir aprendiendo. Aprendimos sobre cómo funcionaba el diseño cliente-maestro-trabajadores, el cual está diseñado para realizar cálculos paralelos distribuyendo tareas entre un proceso maestro y múltiples procesos trabajadores. Este patrón mejora la concurrencia, el rendimiento y la escalabilidad en los sistemas de software.

En cuanto a posibles mejoras que se podrían implementar en el software sería encontrar una manera de distribuir equitativamente no la cantidad de números que cada worker va a evaluar sino el trabajo que cada worker debe hacer para encontrar números perfectos en su rango, ya que para rangos de números más grande, el worker que tiene asignado este rango debe hacer mucho más esfuerzo y gasta más energía, tiempo y recursos en realizar muchas divisiones para posteriormente sumarlas y tan sólo verificar si los números son perfectos.

Link al repositorio en GitHub: https://github.com/SimonGarcia01/ProyectoCompunet1

Referencias

- https://www.spiceworks.com/tech/cloud/articles/what-is-distributed-computing/
- https://prezi.com/rlt7lzvk0z8m/arquitectura-de-sistemas-distribuidos/
- https://www.edenred.es/blog/eficiencia-eficacia-y-efectividad-diferencias-y-calculo/
- https://java-design-patterns.com/patterns/master-worker/
- Proyecto “09_ice_gradle” localizada en el repositorio de GitHub “Computacion-I-2025-1”del profesor Nicolás Salazar.