Proyecto Final Computacion en Internet I

Por: Santiago Zapata Rodriguez, Jose Miguel Armas y Juan Pablo Sinisterra

Descripción del Problema

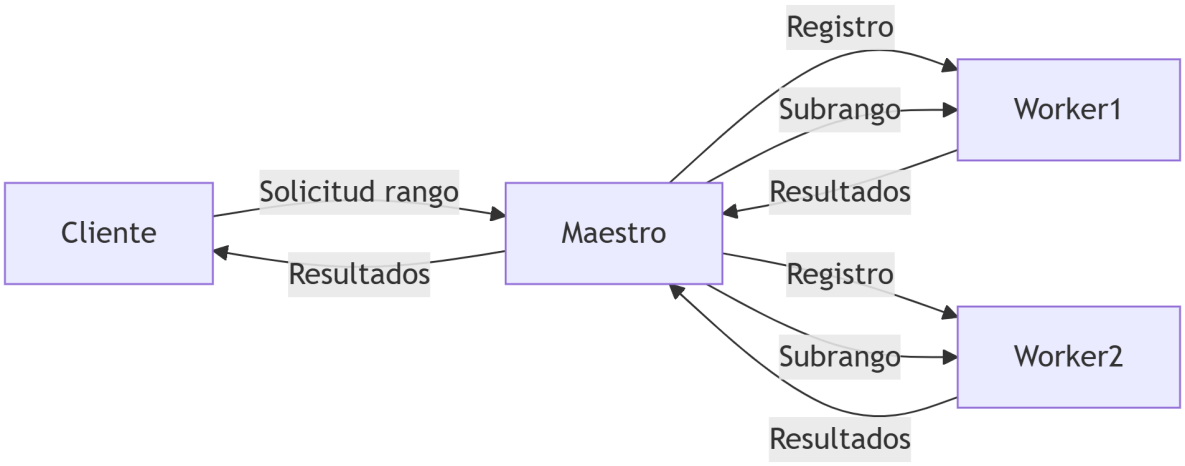
El problema consiste en diseñar e implementar un sistema distribuido capaz de encontrar números perfectos dentro de un rango dado, optimizando el uso de recursos mediante la paralelización. Dado que la búsqueda de números perfectos es computacionalmente costosa (complejidad **O(n√n)**), se propone una arquitectura basada en el modelo Cliente-Maestro-Trabajadores sobre el middleware ICE (Internet Communications Engine), utilizando llamadas asíncronas para lograr eficiencia y escalabilidad. El cliente envía la solicitud del rango, el maestro divide la carga entre múltiples trabajadores, y estos calculan de forma concurrente los números perfectos en sus respectivos subrangos, devolviendo los resultados para ser consolidados y presentados al usuario.

Análisis del Algoritmo Utilizado

El algoritmo utilizado para verificar si un número n es perfecto implica iterar desde 1 hasta n/2, sumando todos los divisores positivos. Esta operación tiene una complejidad **O(n√n)** si se optimiza revisando hasta √n y sumando divisores en pares.

Arquitectura general del sistema distribuido

El sistema implementa una arquitectura distribuida Cliente-Maestro-Trabajadores usando ICE (Internet Communications Engine) como middleware. El Cliente solicita encontrar los números perfectos dentro de un rango. El Maestro divide ese rango en subrangos y los asigna a múltiples Trabajadores, quienes calculan los números perfectos y devuelven los resultados al Maestro, quien consolida y responde al Cliente.

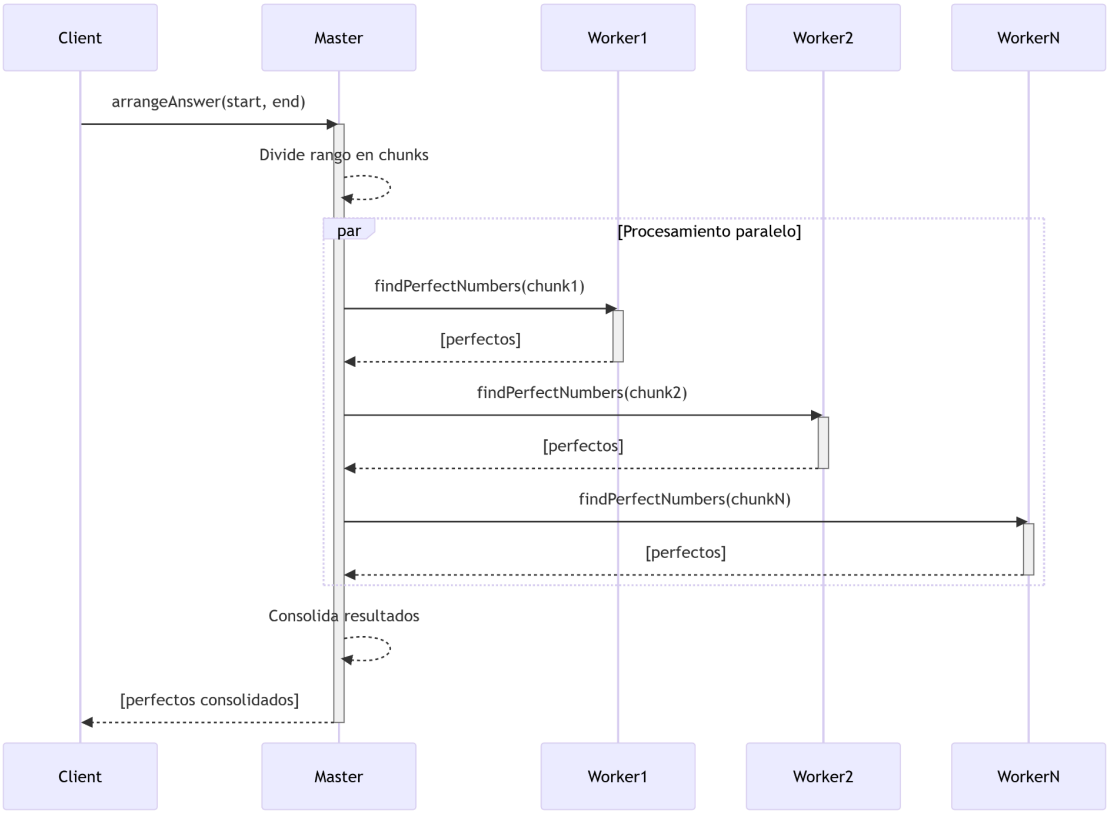


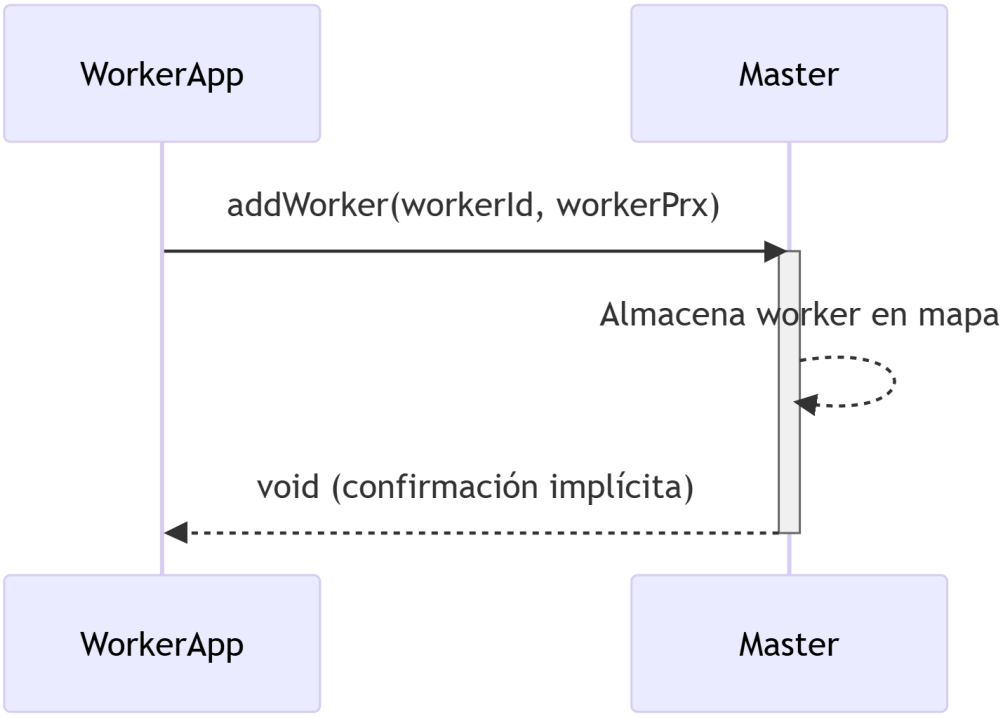
Diseño Cliente-Maestro-Trabajador ICE

El sistema sigue un diseño cliente-maestro-trabajador basado en ICE, donde las interfaces Slice definen los contratos de comunicación:

- El Worker expone **findPerfectNumbers(start, end)** para procesar subrangos.
- El Master gestiona el registro dinámico de workers **(addWorker)** y la distribución de tareas **(arrangeAnswer)**.

Los workers se registran automáticamente al iniciar, estableciendo conexiones persistentes mediante proxies Ice, y el maestro coordina el trabajo dividiendo el rango solicitado por el cliente en subrangos equilibrados, que asigna a los workers disponibles mediante llamadas asíncronas implementadas con Futures de Java para evitar bloqueos. El sistema maneja errores con timeouts (de 5 minutos por worker), reconexiones automáticas en fallos y tolerancia a fallos mediante un ConcurrentHashMap que rastrea a los workers activos. Los resultados parciales se consolidan en un ArrayList thread-safe antes de enviarse al cliente, optimizando el procesamiento con algoritmos eficientes.


Client
Master
Worker1
Worker2
WorkerN
arrangeAnswer(start, end)
Divide rango en chunks
par
[Procesamiento paralelo]
findPerfectNumbers(chunk1)
[perfectos]
findPerfectNumbers(chunk2)
[perfectos]
findPerfectNumbers(chunkN)
[perfectos]
Consolida resultados
[perfectos consolidados]
Client
Master
Worker1
Worker2
WorkerN


WorkerApp
Master
addWorker(workerId, workerPrx)
Almacena worker en mapa
void (confirmación implícita)
WorkerApp
Master

Explicación del mecanismo de distribución del rango y coordinación

El maestro implementa un algoritmo estático de división de rangos que calcula subrangos de tamaño similar. Por ejemplo, para el rango [1, 100] con 3 workers: Worker1 procesa [1,33], Worker2 [34,66], y Worker3 [67,100]. La gestión de concurrencia utiliza un ExecutorService con pool de hilos fijo, donde cada subrango se procesa en un hilo independiente. Los resultados parciales se recopilan mediante Futures, garantizando la integridad de los datos finales. Este enfoque asegura carga equilibrada en entornos homogéneos, aunque podría presentar desequilibrios con rangos que contengan concentraciones desiguales de números perfectos.

Resultados experimentales y análisis de rendimiento

Prueba	Rango de Números	Número de Workers	Tiempo	Resultados
1	1-10,000	2	Primer Intento: 120.122 sec Segundo Intento: 0.009 sec	6, 28, 496, 8128
2	1-10,000	4	Primer Intento: 60.038 sec Segundo Intento: 0.006 sec	6, 28, 496, 8128
3	1-10,000	8	Primer Intento: 120.175 sec Segundo Intento: 60.016 sec	6, 28, 496, 8128
4	1-100,000	2	0.034 sec	6, 28, 496, 8128
5	1-100,000	4	0.020 sec	6, 28, 496, 8128
6	1-100,000	8	0.025 sec	6, 28, 496, 8128
7	1-1,000,000	2	0.498 sec	6, 28, 496, 8128

8	1-1,000,000	4	0.267 sec	6, 28, 496, 8128
9	1-1,000,000	8	0.146 sec	6, 28, 496, 8128
10	1-35,000,000	8	27.027 sec	6, 28, 496, 8128, 33.550.336

Análisis de los Resultados

1. ***Primeras Tres Pruebas (Rango 1-10,000):***
    - ***a.*** Se observa una disminución en el tiempo al aumentar el número de trabajadores de 2 a 4 (de 120.122 segundos a 60.038 segundos). Sin embargo, al aumentar a 8 trabajadores, el tiempo vuelve a subir a 120.175 segundos. Esto se debe a la sobrecarga de gestión de más hilos y posiblemente a la naturaleza del rango (que es pequeño) y al tiempo de inicio de los workers.
    - ***b.*** Se nota una gran diferencia entre el primer y segundo intento en las primeras tres pruebas. Esto se debe a que el primer intento incluye el tiempo de inicio de los trabajadores y la conexión con el maestro, mientras que en el segundo intento los trabajadores ya están en funcionamiento.

2. ***Pruebas 4 a 6 (Rango 1-100,000):***
    - ***a.*** Los tiempos son muy bajos (fracciones de segundo) y no se observa una mejora significativa al aumentar el número de trabajadores. Esto puede deberse a que el rango es manejable incluso con un solo trabajador, y la sobrecarga de la distribución y consolidación de resultados podría estar afectando.

3. ***Pruebas 7 a 9 (Rango 1-1,000,000):***
    - ***a.*** Aquí se observa una mejora en el tiempo a medida que se aumenta el número de trabajadores: 0.498 segundos (2 workers), 0.267 segundos (4 workers) y 0.146 segundos (8 workers). Esto indica que para rangos más grandes, el paralelismo es efectivo.

4. ***Prueba 10 (Rango 1-35,000,000):***

a. Con 8 trabajadores, el tiempo es de 27.027 segundos. Se encontró el quinto número perfecto: 33,550,336. Este resultado demuestra que el sistema es capaz de manejar rangos muy grandes de manera eficiente.

Eficiencia y Escalabilidad

Escalabilidad: El sistema escala bien para rangos grandes, como se observa en las pruebas 7 a 10. Al aumentar el número de trabajadores, el tiempo de procesamiento disminuye.

Eficiencia: Para rangos pequeños, la sobrecarga de la distribución y la gestión de trabajadores puede ser mayor que el beneficio del paralelismo (pruebas 1-3). Sin embargo, para rangos grandes, la eficiencia es clara.

Conclusiones y posibles mejoras

El sistema logra distribuir efectivamente el trabajo para encontrar números perfectos en un rango, aprovechando las capacidades del modelo Cliente-Maestro-Trabajadores y las comunicaciones asíncronas de ICE. Las mejoras posibles incluyen:

- Balance de carga dinámica (trabajadores más rápidos podrían recibir más trabajo).
- Tolerancia a fallos con reintentos.
- Interfaz gráfica para el cliente.
- Soporte para múltiples clientes concurrentes.