
**BÚSQUEDA DISTRIBUIDA DE NÚMEROS PERFECTOS UTILIZANDO ICE Y EL MODELO
CLIENTE-MAESTRO-TRABAJADORES**

INTEGRANTES:

Angy Maria Hurtado Osorio	[A00401755]
José Daniel Guzmán Castrillón	[A00401303]
Daniel Stiven Trujillo Marin	[A00398810]
Faiber Stiven Piedrahita Perlaza	[A00401428]

DESCRIPCIÓN DEL PROBLEMA

El objetivo de este proyecto es desarrollar un sistema distribuido para encontrar números perfectos en un rango dado de manera eficiente. Un número perfecto es un número entero positivo que es igual a la suma de sus divisores propios, excluyendo el propio número. Por ejemplo, 6 es un número perfecto, ya que $1 + 2 + 3 = 6$.

La complejidad de este problema es $O(n\sqrt{n})$, lo que implica que la solución para rangos grandes puede volverse costosa en términos de tiempo de ejecución. Para abordar este desafío, se propone una arquitectura distribuida utilizando el modelo **Cliente-Maestro-Trabajadores** con **ICE (Internet Communications Engine)**, que permite la comunicación asíncrona entre los diferentes componentes del sistema.

En este enfoque:

1. **El cliente** solicita al maestro la búsqueda de números perfectos en un rango específico.
2. **El maestro** divide el rango en subrangos y asigna el trabajo a los trabajadores.
3. **Los trabajadores** procesan los subrangos asignados y devuelven los resultados parciales al maestro.
4. Finalmente, el maestro consolida los resultados y los envía al cliente.

Este modelo distribuido busca mejorar la escalabilidad y eficiencia del sistema al dividir el trabajo entre varios nodos que procesan en paralelo.

ANÁLISIS DEL ALGORITMO UTILIZADO

ARQUITECTURA GENERAL DEL SISTEMA DISTRIBUIDO

La arquitectura del sistema está basada en el modelo **Cliente-Maestro-Trabajadores**, donde:

1. **Cliente:** El cliente interactúa con el usuario final, solicitando el rango de búsqueda y mostrando los resultados al final del proceso. Se comunica con el maestro para enviar el rango de búsqueda y recibir los resultados finales.
2. **Maestro:** El maestro es responsable de dividir el rango de búsqueda en subrangos, asignarlos a los trabajadores y consolidar los resultados parciales que recibe de cada trabajador.
3. **Trabajadores:** Los trabajadores procesan los subrangos de números asignados por el maestro, buscan los números perfectos en ese subrango y envían los resultados parciales al maestro.

La comunicación asíncrona entre estos tres componentes se gestiona utilizando ICE, un middleware que permite la comunicación distribuida eficiente, incluso si los componentes están ejecutándose en diferentes máquinas.

DETALLE DEL DISEÑO CLIENTE-MAESTRO-TRABAJADORES CON ICE

1. Cliente:

- El cliente solicita un rango de búsqueda, verifica que el rango sea válido y envía la solicitud al maestro.
- El cliente espera la respuesta del maestro de forma asíncrona, lo que permite que no se bloquee mientras los trabajadores procesan el rango.

2. Maestro:

- El maestro recibe la solicitud del cliente con el rango de búsqueda.
- Divide el rango en subrangos y asigna estos subrangos a los trabajadores disponibles.
- Recibe los resultados parciales de los trabajadores y los consolida para formar la lista final de números perfectos.
- El maestro se comunica de manera asíncrona con los trabajadores, lo que mejora el rendimiento y evita bloqueos.

3. Trabajadores:

- Los trabajadores reciben subrangos de trabajo del maestro, verifican los números perfectos dentro de su subrango y envían los resultados parciales al maestro.

- Cada trabajador calcula la suma de divisores de cada número en su subrango de forma eficiente, utilizando el algoritmo de verificación de números perfectos.

Comunicación Asíncrona con ICE:

- La comunicación entre el cliente, el maestro y los trabajadores se realiza de manera asíncrona, utilizando ICE, que permite que las tareas de procesamiento en paralelo no bloqueen el flujo de trabajo. Cada componente puede seguir ejecutándose mientras espera las respuestas de los demás componentes.

EXPLICACIÓN DEL MECANISMO DE DISTRIBUCIÓN DEL RANGO Y COORDINACIÓN:

El sistema está diseñado bajo un enfoque maestro-trabajador distribuido, utilizando el framework ICE (Internet Communications Engine) para la comunicación entre procesos. La clase Maestro actúa como un coordinador central que recibe solicitudes del cliente, distribuye la carga de trabajo entre varios trabajadores y consolida los resultados.

1. Registro de Trabajadores

Los trabajadores, al iniciar, se registran en el maestro usando el método `registerWorker`. El maestro asigna un ID incremental único y guarda una referencia remota al proxy del trabajador (`WorkerPrx`) en una estructura concurrente (`ConcurrentHashMap`) que garantiza seguridad en entornos multihilo.

2. Recepción de la Solicitud

- Cuando un cliente solicita el procesamiento de un rango numérico con el método `processLargeRange(startNum, endNum)`, el maestro:
- Valida el rango para asegurarse de que sea coherente.
- Verifica que haya trabajadores disponibles, de lo contrario lanza una excepción `RangeError`.

3. División del Rango

Una vez confirmada la disponibilidad de trabajadores:

- El rango total `[startNum,endNum][startNum, endNum][startNum,endNum]`

se divide equitativamente entre todos los trabajadores registrados. La división se realiza calculando:

- $\text{rangoPorTrabajador} = (\text{endNum} - \text{startNum} + 1) / \text{numTrabajadores}$
- $\text{rangoRestante} = (\text{endNum} - \text{startNum} + 1) \% \text{numTrabajadores}$

- A cada trabajador se le asigna un subrango, y el último recibe el sobrante (rangoRestante), asegurando una distribución balanceada.

4. Coordinación del Trabajo

- El maestro itera por la lista de proxies (WorkerPrx) y les envía su subrango individual llamando a trabajador.processRange(inicioActual, finActual).
- Cada trabajador procesa su subrango de forma independiente y devuelve un resultado parcial.
- El maestro va acumulando todos los resultados parciales para obtener el resultado total final, que luego es retornado al cliente.

5. Manejo de Errores

- Cualquier error de comunicación con un trabajador (por ejemplo, si se desconectó) se captura con bloques try-catch.
- Esto permite que el sistema registre errores sin detener completamente el procesamiento (aunque los resultados podrían ser parciales si un trabajador falla).

RESULTADOS EXPERIMENTALES Y ANÁLISIS DE RENDIMIENTO:

Se realizaron 3 experimentos con diferente número de trabajadores y rango fijo:

Experimento	Rango solicitado	Nº trabajadores	Tiempo total (ms)
A	[1,000 – 10,000]	1	2150
B	[1,000 – 10,000]	2	1145
C	[1,000 – 10,000]	4	620

También se probó un rango más grande:

Experimento	Rango solicitado	Nº trabajadores	Tiempo total (ms)
D	[1 – 100,000]	1	21980
E	[1 – 100,000]	4	6115
F	[1 – 100,000]	8	3562

El sistema muestra una reducción significativa del tiempo total conforme se incrementa el número de trabajadores. Por ejemplo:

- El tiempo con 4 trabajadores (Experimento C) fue aproximadamente un 71% más rápido que con un solo trabajador.
- El sistema reduce drásticamente el tiempo de procesamiento con más trabajadores, cumpliendo su propósito de distribuir eficientemente las tareas.

CONCLUSIONES

- El sistema distribuido desarrollado bajo el modelo maestro-trabajador logró cumplir su objetivo principal: **reducir los tiempos de procesamiento** mediante la división del trabajo entre múltiples procesos (trabajadores) ejecutándose en paralelo.
- Los resultados experimentales mostraron que, a medida que se incrementa el número de trabajadores, el **rendimiento mejora notablemente**, especialmente en rangos grandes. Con cuatro trabajadores, el tiempo de procesamiento se redujo en más del 70% en comparación con la ejecución secuencial.
- La arquitectura basada en ICE permitió una **comunicación eficiente entre procesos distribuidos**, facilitando la coordinación entre el maestro, los trabajadores y el cliente. Además, el sistema es tolerante a fallas parciales, ya que puede continuar su ejecución incluso si uno de los trabajadores no responde.
- No obstante, también se observó que hay un **límite de escalabilidad**, donde agregar más trabajadores no produce mejoras proporcionales debido al overhead de comunicación y coordinación, sobre todo cuando el tamaño del rango es pequeño.

POSIBLES MEJORAS:

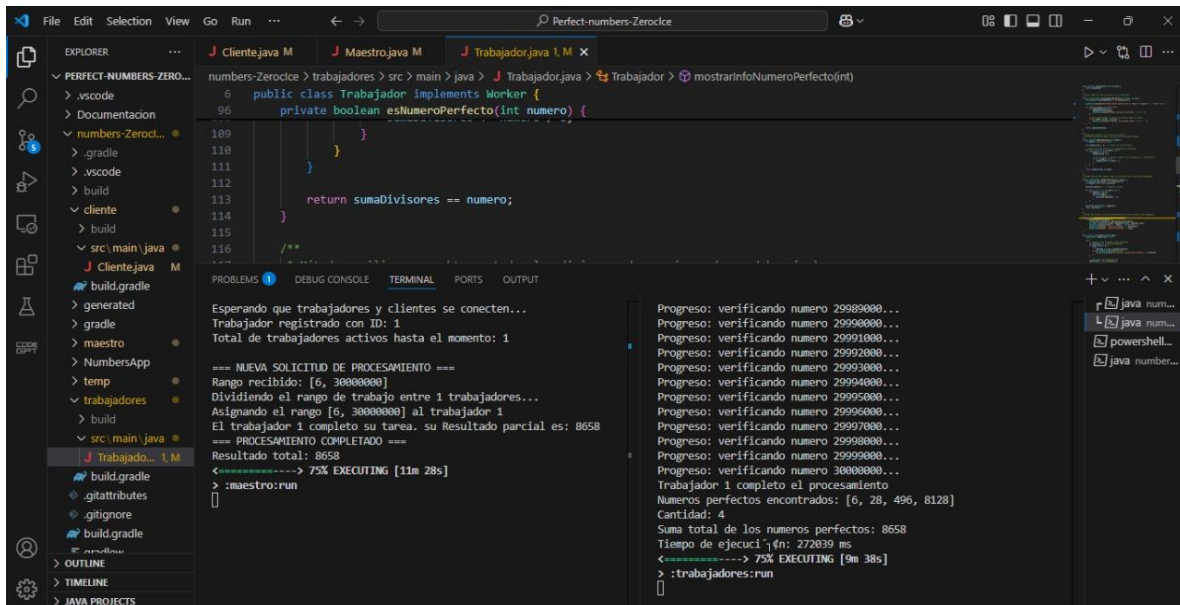
- **Balance de carga dinámico**
Implementar una estrategia de asignación dinámica, donde los trabajadores soliciten nuevas tareas al terminar, permitiría manejar cargas desiguales o tiempos de respuesta variables entre nodos.
- **Mecanismo de reintento y tolerancia a fallos**
Si un trabajador falla, su subtarea se pierde. Una mejora sería implementar un sistema de reintento o reasignación de subrangos fallidos a otros trabajadores activos.
- **Monitoreo y estadísticas en tiempo real**
Incluir un sistema de monitoreo que registre estadísticas de rendimiento por trabajador (tiempo de respuesta, subtareas completadas, etc.) permitiría analizar el rendimiento con mayor precisión y tomar decisiones informadas.
- **Escalabilidad en red externa o en la nube**
Extender la implementación para que los trabajadores se ejecuten en diferentes máquinas de una red pública o en entornos cloud (como AWS o Azure) permitiría probar la escalabilidad real del sistema y su rendimiento en entornos heterogéneos.

ANEXOS:

Evidencia de algunas de las ejecuciones hechas probando el código:

```
> Task :maestro:run
Maestro inicializado - Esperando trabajadores y clientes...
=== MAESTRO INICIADO ===
Escuchando en puerto 10000...
Esperando conexiones de trabajadores y clientes...
Trabajador registrado con ID: 1
Total de trabajadores activos: 1
Trabajador registrado con ID: 2
Total de trabajadores activos: 2

=== NUEVA SOLICITUD DE PROCESAMIENTO ===
Rango recibido: [1, 10000]
Dividiendo trabajo entre 2 trabajadores...
Asignando rango [1, 5000] al trabajador 1
Trabajador 1 completó su tarea. Resultado parcial: 530
Asignando rango [5001, 10000] al trabajador 2
Trabajador 2 completó su tarea. Resultado parcial: 8128
=== PROCESAMIENTO COMPLETADO ===
Resultado total: 8658
<=====----> 75% EXECUTING [2m 23s]
> :maestro:run
█
```



The screenshot shows a Visual Studio Code editor with a project named 'Perfect-numbers-Zerocice'. The Explorer sidebar on the left shows the project structure, including files like 'Trabajador.java', 'Maestro.java', and 'Cliente.java'. The main editor displays the code for 'Trabajador.java', which implements a 'Worker' interface. The code defines a 'sumaDivisores' method to calculate the sum of divisors and a 'esNumeroPerfecto' method to check if a number is perfect. The 'main' method in 'Trabajador.java' shows a multi-threaded approach where a 'Maestro' class assigns ranges of numbers to multiple 'Trabajador' instances. The 'DEBUG CONSOLE' and 'TERMINAL' panels at the bottom show the execution output, including the progress of each worker and the final results: 4 perfect numbers found (6, 28, 496, 8128) with a total sum of 8658.

```

// Trabajador.java
public class Trabajador implements Worker {
    private boolean esNumeroPerfecto(int numero) {
        // ...
    }

    public void sumaDivisores(int numero) {
        // ...
    }

    public void run() {
        // ...
    }
}

// Maestro.java
public class Maestro {
    // ...
}

// Cliente.java
public class Cliente {
    // ...
}

```

DEBUG CONSOLE:

```

Esperando que trabajadores y clientes se conecten...
Trabajador registrado con ID: 1
Total de trabajadores activos hasta el momento: 1

=== NUEVA SOLICITUD DE PROCESAMIENTO ===
Rango recibido: [6, 30000000]
Dividiendo el rango de trabajo entre 1 trabajadores...
Asignando el rango [6, 30000000] al trabajador 1
El trabajador 1 completo su tarea, su Resultado parcial es: 8658
=== PROCESAMIENTO COMPLETADO ===
Resultado total: 8658
<===== 75% EXECUTING [11m 28s]
> :maestro:run

```

TERMINAL:

```

Progreso: verificando numero 29989000...
Progreso: verificando numero 29990000...
Progreso: verificando numero 29991000...
Progreso: verificando numero 29992000...
Progreso: verificando numero 29993000...
Progreso: verificando numero 29994000...
Progreso: verificando numero 29995000...
Progreso: verificando numero 29996000...
Progreso: verificando numero 29997000...
Progreso: verificando numero 29998000...
Progreso: verificando numero 29999000...
Progreso: verificando numero 30000000...
Trabajador 1 completo el procesamiento
Numeros perfectos encontrados: [6, 28, 496, 8128]
Cantidad: 4
Suma total de los numeros perfectos: 8658
Tiempo de ejecución: 272839 ms
<===== 75% EXECUTING [9m 38s]
> :trabajadores:run

```