INFORME BÚSQUEDA DISTRIBUIDA

DE NÚMEROS PERFECTOS

INTEGRANTES:

Valentina Tobar Gómez - A00401749

Simón Reyes - A00400880

Melissa Hurtado - A00401116

Santiago Grajales - A00402018

PROFESOR

Nicolas Javier Salazar Echeverry

COMPUTACIÓN EN INTERNET I

CALI, VALLE DEL CAUCA

UNIVERSIDAD ICESI

10 JUNIO 2025

1. Descripción del Problema y Análisis del Algoritmo

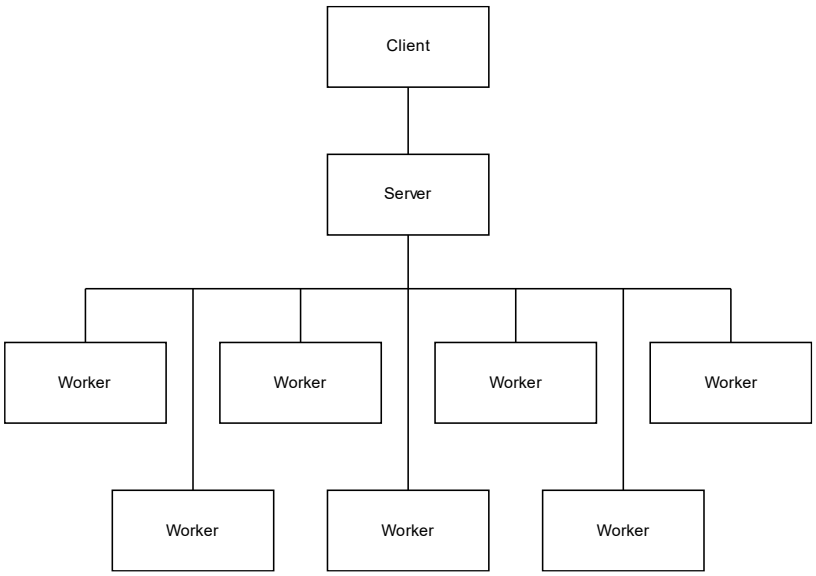
Este proyecto tiene como objetivo encontrar números perfectos, aquellos que son iguales a la suma de sus divisores propios positivos, dentro de un rango definido por el usuario. Dado que este cálculo es computacionalmente costoso (complejidad $O(n\sqrt{n})$), se plantea una solución distribuida basada en ICE (ZeroC), utilizando una arquitectura cliente-maestro-trabajadores. En esta, el cliente envía la solicitud del rango, el maestro lo divide en subrangos y asigna cada uno a diferentes trabajadores, quienes procesan los datos en paralelo y devuelven los resultados. El sistema debe ser eficiente, asíncrono, tolerante a fallos y fácilmente escalable con respecto al número de trabajadores involucrados.

2. Arquitectura del Sistema Distribuido

El sistema está compuesto por cuatro módulos principales que interactúan a través de comunicaciones definidas mediante ICE:

- **Cliente (Client)**: Funciona como la interfaz de entrada que permite al usuario definir el rango de búsqueda, la cantidad de trabajadores que realizaran la búsqueda, la forma en que quieren que se comporten los trabajadores (sincrónico o asincrónico) y recibe los resultados.
- **Maestro (Server)**: Es el encargado de dividir el trabajo, coordinar trabajadores y consolidar los resultados.
- **Trabajadores (Worker)**: Ejecutan la búsqueda de números perfectos en subrangos asignados por el maestro y devuelve los resultados obtenidos de esos subrangos.
- **Interfaces Slice**: Definen la comunicación entre módulos usando ICE, de los cuales se definen 3 interfaces (ClientCallback, Coordinador y Worker).

A continuación, se presenta un diagrama que ilustra esta arquitectura:



3. Diseño Cliente-Maestro-Trabajadores con ICE

Como se mencionó anteriormente, el sistema se basa en el modelo Cliente-Maestro-Trabajadores, el cual se implementó con ICE para facilitar la comunicación entre los procesos distribuidos. Con este diseño es posible paralelizar la búsqueda de números perfectos, y está compuesto por los siguientes componentes:

Cliente:

El Cliente solicita al usuario un valor máximo y considera el rango de búsqueda desde 0 hasta ese valor. Además, solicita la cantidad de trabajadores que se encargarán de la tarea y el tipo de ejecución (asíncrona o síncrona), según las necesidades del usuario. Se comunica con el Maestro mediante una llamada asíncrona para enviar los parámetros de la solicitud.

Posteriormente, espera la respuesta del Maestro y, una vez recibida, muestra al usuario los números perfectos encontrados junto con el tiempo total de ejecución.

Maestro:

El Maestro recibe la solicitud del Cliente, incluyendo el rango de búsqueda, la cantidad de trabajadores y el modo de ejecución.

Verifica la disponibilidad de trabajadores y ajusta la cantidad final en función de lo solicitado por el Cliente.

Divide el rango de búsqueda en subrangos equitativos según el número de trabajadores asignados.

Distribuye cada subrango entre los trabajadores mediante llamadas asíncronas o síncronas, según lo especificado.

Recibe los resultados parciales de los trabajadores, los consolida y los envía de regreso al Cliente.

Trabajadores:

Cada Trabajador recibe un subrango asignado por el Maestro, ejecuta el algoritmo para detectar números perfectos dentro de dicho subrango y devuelve los resultados encontrados al Maestro.

4. Mecanismo de Distribución y Coordinación

El Maestro consulta cuántos trabajadores tiene disponibles y asigna la cantidad más adecuada en función de la solicitud del Cliente. Luego, calcula el tamaño de los subrangos dividiendo equitativamente el rango total de búsqueda entre el número de trabajadores asignados. Esta distribución permite paralelizar el procesamiento de forma eficiente, aprovechando los recursos del sistema distribuido.

Una vez determinados los subrangos, el Maestro utiliza llamadas asincrónicas proporcionadas por ICE (Internet Communications Engine) para enviar cada subrango a los trabajadores correspondientes. Además, se emplean hilos en el Maestro para gestionar de manera concurrente el envío de tareas y la espera de resultados, sin bloquear el proceso principal. Esta combinación de asincronía en la comunicación y concurrencia interna permite que la ejecución sea no bloqueante, lo cual mejora el rendimiento general del sistema.

Cada trabajador procesa el subrango que le fue asignado, ejecutando el algoritmo de búsqueda de números perfectos. Al finalizar, devuelve al Maestro una lista con los números perfectos encontrados en su sección del rango.

El Maestro recibe las listas parciales provenientes de todos los trabajadores. Estas listas pueden ser ordenadas, si así se requiere, para facilitar la presentación final. Finalmente, el Maestro consolida los resultados y los envía al Cliente para su visualización.

5. Resultados Experimentales y Análisis de Rendimiento

Para la realización del experimento, requerimos de como mínimo 8 computadores con las mismas especificaciones, además de eso, necesitábamos una red privada en la cual hacerlo. En nuestro caso utilizamos la sala de laboratorio de la universidad, en el cual se realizaron las peticiones y arrojaron los siguientes resultados.

Tabla de resultados experimentales de manera sincrónica

Rango de búsqueda	Nº de trabajadores	Tiempo total (s)	Números perfectos encontrados
0 – 10000	2	0,041	[6, 28, 496, 8128]
0 – 100000	2	3,228	[6, 28, 496, 8128]
0 – 1000000	2	320,667	[6, 28, 496, 8128]
0 – 10000	4	0,055	[6, 28, 496, 8128]
0 – 100000	4	3,78	[6, 28, 496, 8128]
0 – 1000000	4	372,759	[6, 28, 496, 8128]
0 – 10000	6	0,061	[6, 28, 496, 8128]
0 – 100000	6	3,682	[6, 28, 496, 8128]
0 – 1000000	6	361,885	[6, 28, 496, 8128]
0 – 10000	8	0,055	[6, 28, 496, 8128]
0 – 100000	8	2,856	[6, 28, 496, 8128]
0 – 1000000	8	278,078	[6, 28, 496, 8128]

Grafica Tiempo vs Número de trabajadores

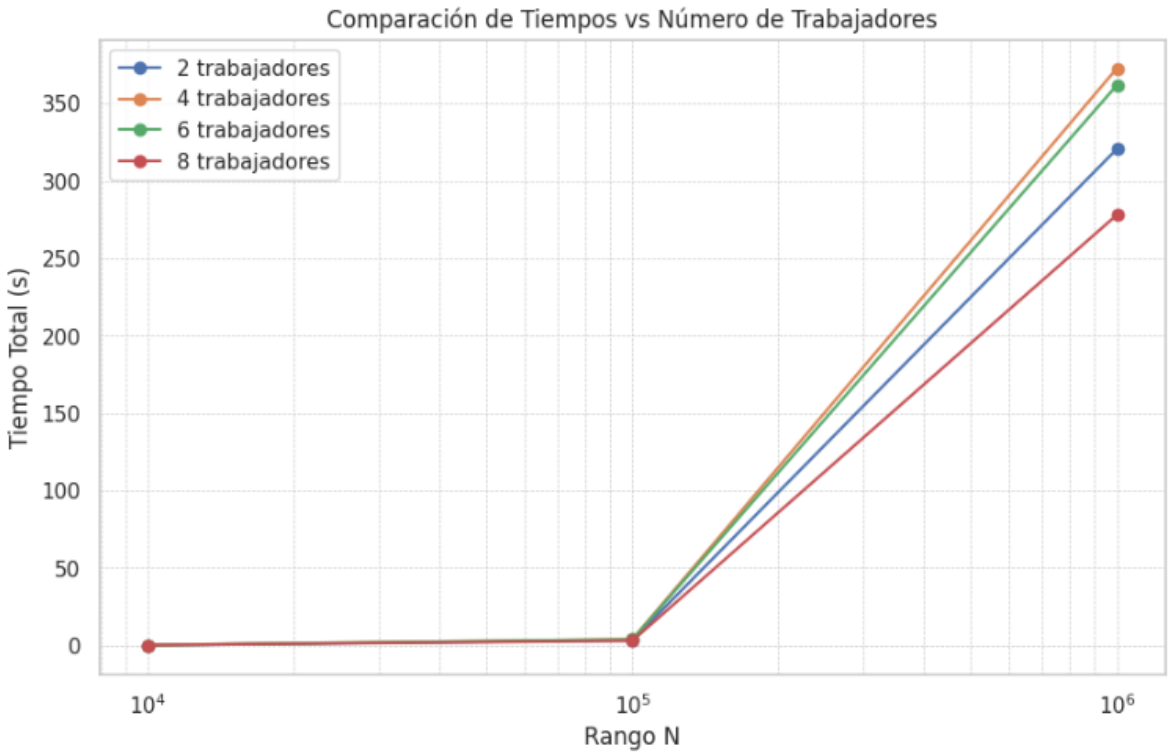
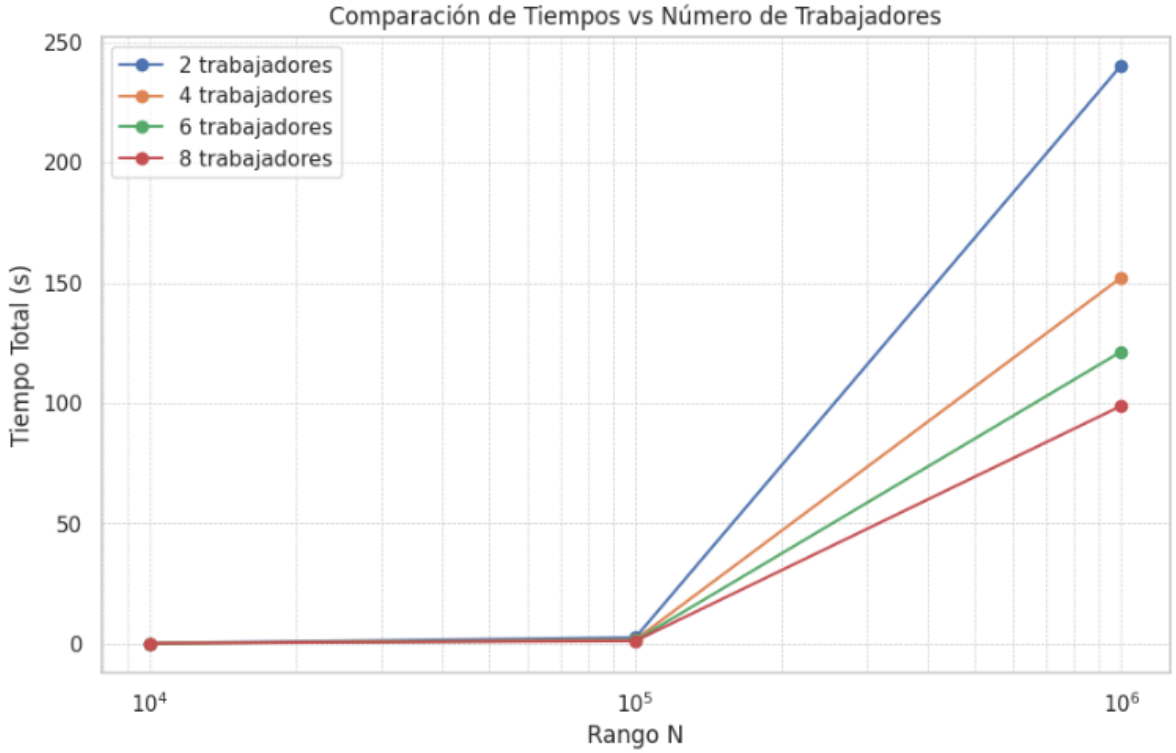


Tabla de resultados experimentales de manera asincrónica

Rango de búsqueda	Nº de trabajadores	Tiempo total (s)	Números perfectos encontrados
0 – 10000	2	0,029	[6, 28, 496, 8128]
0 – 100000	2	2,414	[6, 28, 496, 8128]
0 – 1000000	2	240,427	[6, 28, 496, 8128]
0 – 10000	4	0,021	[6, 28, 496, 8128]
0 – 100000	4	1,526	[6, 28, 496, 8128]
0 – 1000000	4	152,15	[6, 28, 496, 8128]
0 – 10000	6	0,018	[6, 28, 496, 8128]
0 – 100000	6	1,221	[6, 28, 496, 8128]
0 – 1000000	6	121,348	[6, 28, 496, 8128]
0 – 10000	8	0,025	[6, 28, 496, 8128]
0 – 100000	8	0,993	[6, 28, 496, 8128]
0 – 1000000	8	98,753	[6, 28, 496, 8128]

Grafica Tiempo vs Número de trabajadores



Observaciones:

Con base en las tablas de resultados experimentales y gráficas, se pueden observar varias tendencias claras que evidencian las diferencias entre las ejecuciones sincrónica y asincrónica en términos de rendimiento y escalabilidad.

En primer lugar, se aprecia que la implementación asincrónica es más rápida que la sincrónica en todos los escenarios evaluados, independientemente del tamaño del rango o de la cantidad de trabajadores. Por ejemplo, para el rango más amplio (0 – 1,000,000) utilizando 8 trabajadores, la ejecución sincrónica tomó aproximadamente 278 segundos, mientras que la asincrónica redujo ese tiempo a 98 segundos, lo cual representa una mejora del 65% en eficiencia.

Además, conforme se incrementa el número de trabajadores, los tiempos de ejecución disminuyen más drásticamente en la versión asincrónica. Esto indica que el sistema asincrónico escala de manera más efectiva, aprovechando mejor el paralelismo y reduciendo los cuellos de botella generados por operaciones bloqueantes. En cambio, la implementación sincrónica muestra una mejora más limitada al aumentar el número de trabajadores, e incluso en algunos casos los tiempos tienden a incrementarse levemente, lo que presenta un uso menos eficiente de los recursos. En ambas versiones se encontraron los mismos números perfectos, por lo que la rapidez de la versión asincrónica no afecta la precisión del resultado.

Finalmente, al observar las gráficas, podemos concluir que el algoritmo funciona de manera eficiente para tamaños de búsqueda N no superiores a 100,000. En este rango, el tiempo de ejecución es similar independientemente de la cantidad de trabajadores utilizados.

Sin embargo, a partir de N > 100.000, la complejidad algorítmica de encontrar números perfectos comienza a impactar significativamente el tiempo de procesamiento. Es precisamente en este punto donde la arquitectura y el enfoque que implementamos con ICE demuestran su eficacia.

Podemos observar que, tanto en la versión síncrona como en la asincrónica, a medida que aumentamos el número de trabajadores, el tiempo de ejecución disminuye notablemente para tamaños mayores a 100,000. Esto confirma que la paralelización mejora considerablemente el rendimiento y representa una solución efectiva para enfrentar la alta complejidad y el costo computacional del problema.

En resumen, usar la versión asincrónica permite hacer el trabajo mucho más rápido, sobre todo cuando se trata de rangos grandes y se tienen varios trabajadores disponibles. Esto se logra porque el sistema puede repartir mejor las tareas y no necesita esperar a que una termine para empezar otra. Este enfoque permite reducir los tiempos de espera, mejorar la concurrencia y hacer un uso más efectivo del sistema distribuido.

6. Conclusiones y Posibles Mejoras

Conclusiones:

La comparación entre la ejecución sincrónica y la asincrónica demuestra que el uso de asincronismo con ICE, combinado con hilos, mejora significativamente el rendimiento del sistema, especialmente al trabajar con rangos grandes. A medida que se incrementa el número de trabajadores, se reduce el tiempo total de procesamiento en ambas implementaciones, pero la reducción es mucho más notoria en el modelo asincrónico. Esto evidencia que el sistema es escalable y que la combinación de asincronismo y concurrencia permite aprovechar mejor los recursos disponibles.

Además, los resultados obtenidos son consistentes en todas las pruebas: ambos modelos identifican correctamente los números perfectos en cada rango, lo que valida la exactitud del sistema. El uso de hilos fue clave para permitir que el Maestro asignara tareas de manera eficiente a los trabajadores y pudiera esperar los resultados sin bloquear el resto de la ejecución.

Posibles mejoras:

Una posible mejora sería implementar un sistema de balance dinámico de carga. En lugar de repartir los subrangos de forma equitativa desde el principio, se podría evaluar en tiempo real el progreso de cada trabajador y asignar más trabajo a los que terminen antes, evitando tiempos muertos.

También sería útil agregar una funcionalidad que permita guardar los resultados obtenidos en una base de datos o archivo. Esto facilitaría el análisis posterior y evitaría repetir los cálculos en futuras ejecuciones.

Otra mejora interesante sería aplicar patrones de diseño que favorezcan la extensibilidad y el mantenimiento del sistema. Por ejemplo, el patrón Strategy permitiría encapsular distintos algoritmos de búsqueda (como números perfectos, primos o abundantes) y seleccionarlos dinámicamente según la necesidad del usuario. También se podría emplear el patrón Factory para crear trabajadores con distintos comportamientos o configuraciones, y el patrón Observer para notificar al Cliente sobre el progreso o finalización de las tareas. El uso adecuado de estos patrones no solo haría el sistema más flexible, sino también más organizado y fácil de escalar a futuro.

Finalmente, se podría ampliar el sistema para incluir otros tipos de búsquedas, como números primos o abundantes, y optimizar aún más el algoritmo de búsqueda de números perfectos para reducir los tiempos de ejecución, especialmente en rangos muy extensos.