IMPLEMENTACIÓN DE HILOS VIRTUALES PARA MEJORAR LA VELOCIDAD DE PROCESAMIENTO USANDO JAVA 21

IMPLEMENTING VIRTUAL THREADS TO IMPROVE PROCESSING SPEED USING JAVA 21

Berrio Huamani Miguel Angel

Universidad Nacional Tecnológica de Lima Sur, Lima, Perú

RESUMEN

La velocidad de respuesta en las aplicaciones es un factor muy importante a considerar, en ese sentido es importante considerar que las aplicaciones actualmente tienen muchos desafíos que considerar para poder obtener una buena velocidad de respuesta con la limitación de recursos con las que contamos. Actualmente una solución que se ha brindado a fin de mejorar esto, es la programación paralela con ello se puede dividir una tarea en diferentes hilos de ejecución. Así también es importante indicar que la implementación de la programación paralela depende mucho de la tecnología usada, en así que Java en su versión 21 nos presenta una nueva interfaz con la creación de hilos virtuales los cuales rompen la limitante de la cantidad de núcleos del procesador con lo cual solo podíamos crear esa cantidad como un máximo de hilos disponibles, con ello se busca que tanto la velocidad, así como el rendimiento de los servicios tengan una mejora considerable.

Palabras Clave: Velocidad de respuesta, Programación paralela, Java 21, Hilos virtuales

ABSTRACT

The speed of response in applications is a very important factor to consider; in that sense, it is important to consider that applications currently have many challenges to consider in order obtaining a good speed of response with the limited resources we have. Currently, a solution that has been offered in order to improve this is parallel programming, which allows a task to be divided into different execution threads. So it is also important to indicate that the implementation of parallel programming depends a lot on the technology used, so Java in its version 21 presents a new interface with the creation of virtual threads which break the limitation of the number of processor cores with which we could only create that amount as a maximum of available threads, thereby seeking that both the speed and performance of the services have a considerable improvement.

Keywords: Speed of response, Parallel programming, Java 21, Virtual Threads

INTRODUCCIÓN

El modelo secuencial asume que solamente una operación puede ejecutarse a la vez, en un computador con un solo procesador no existe manera de poder mejorar los procesos. No obstante, debido a la evolución tecnología se cuentan actualmente con procesadores multinúcleo, donde cada uno de estos puede ejecutarse de forma independiente. Es con esto que surgió el paradigma de la programación paralela es un nuevo modelo en el cual las operaciones secuenciales son más pequeñas y se ejecutan en paralelo (KhanAcademy, s.f.). Este nuevo paradigma llegó para mejorar el rendimiento de las aplicaciones, no obstante, presenta nuevos retos que resolver entre ellos los procesos seguros o no bloqueantes, los cuales permiten compartir un dato entre diferentes subprogramas paralelos que pueden realizar operaciones sobre el mismo sin afectar la integridad de los datos. En este sentido los hilos o *threads* son los encargados de realizar este tipo de operaciones de manera simultánea, en tecnologías basadas en Java se contaba con la limitante del hardware puesto que la cantidad de hilos máximos eran los disponibles por el procesador a través del sistema operativo. No obstante Java en su versión 21 lanzada el 19/09/2023 entregó una nueva interfaz de hilos o *threads* virtuales con lo cual busca superar la limitante del procesador (Java, s.f.), creando hilos en memoria los cuales pueden extender la cantidad de tareas que se pueden realizar de manera simultánea a través de esta interfaz Java optimiza los hilos disponibles por parte del sistema operativo, con esto logramos obtener mejores tiempos de ejecución sin afectar el rendimiento general de nuestra aplicación, además de poder generar aplicaciones mucho más rápidas sin la necesidad de tener un escalamiento vertical.

MATERIALES Y MÉTODOS

Programación paralela

Este es un nuevo paradigma de programación en la cual cada tarea es ejecutada por un procesador separado, sin necesidad de estos afectarse entre ellos. Cada etapa contribuye al problema general y transmite o continua la ejecución por separar a fin de poder concluir próximas etapas, en ese sentido no todos los tipos de problemas pueden ser abordados por este paradigma. (Saez, Piccoli, Printista, & Gallard, 2003).

Según la tendencia actual, se espera que los procesadores continúen la mejora en el multi-núcleo tanto en la capacidad como en la cantidad, esto puesto que el propio mercado exige tener una alta capacidad de procesamiento. (Ortiz, 2020)

La programación paralela se utiliza para poder resolver problemas en los cuales los recursos en un solo computador o hilo no son suficientes. Con ello buscamos disminuir el tiempo de procesamiento mediante la distribución o separación de tareas entre los diferentes procesadores disponibles. (Vásquez, Valdez, Campos, Campos, & Hernández, 2014)

Un programa paralelo es un tipo de programadas concurrentes el cual está diseñado para ejecutarse en diferentes procesadores de manera segura. Así también existen programadas distribuidos que buscan poder ejecutarse en una red de procesadores autónomos los cuales

no necesariamente comparten recursos, tales como la memoria. (Palma, Garrido, Sánchez, & Quesada, 2003)

Patrón de diseño *Iterator*

Iterator es un patrón de diseño de tipo comportamiento el cual nos permite recorrer diferentes elementos de una colección o arreglo sin necesidad de exponer o representarse a través de una interfaz como una lista o un iterador. (Guru, Iterator, s.f.)

Patrón de diseño *Observer*

El patrón de diseño *Observer* es un patrón de comportamiento que define un mecanismo de suscripción, con el fin de poder notificar a diferentes objetos sobre cualquier cambio o evento que afecte al objeto observado. (Guru, Observer, s.f.)

Procesos seguros

La seguridad de los hilos en Java es muy importante, esto se inicia cuando el entorno empezó a usar multihilos, los cuales pueden compartir la instancia de un objeto y esto puede generar la inconsistencia de los datos a través de subprocesos compartidos que puedan leer y actualizar estos datos. El origen de esta posible inconsistencia se debe a que los procesos requieren de tres pasos, los cuales son leer los valores, realizar operaciones sobre estos para actualizar y posteriormente actualizar estos valores en la variable que hace referencia al espacio de memoria donde se almacena la misma; la razón por la cual esto puede ser un problema en la programación paralela es que existen varios subprocesos que intentan realizar estas operaciones en los datos compartidos. (Oracle, Multithreaded Programming Guide, s.f.)

En Java existe el termino sincronización el cual verifica que los datos sean sincronizados para ser ejecutados en un solo hilo a la vez y así evitar la inconsistencia de datos a través de la ejecución de los mismos, esto se realiza a través de la palabra reservada *synchronized* la cual brinda este proceso a toda clase de tipo de objeto. (DigitalOcean, Seguridad de subprocesos en Java, 22)

En base a ello se lanzaron APIS o interfaces que buscan solucionar la programación asincrónica con flujos de datos observables, estas se soportan fuertemente en patrones *Observer* o *Iterator*, así como la programación funcional, un claro ejemplo es ReactiveX que brinda muchas facilidades de soporte en simultaneidad tanto en procesamiento, así como en el manejo de errores asincrónicos, lo cual brinda más soporte a los procesos seguros y evitar la inconsistencia de datos. (ReactiveX, s.f.)

Existen otras solucionas enfocadas a entornos propios de algunos lenguajes como Java, en ese sentido tenemos Project Reactor el cual es una biblioteca reactiva la cual permite crear aplicaciones sin bloquear la Java Virtual Machine (JVM), esta biblioteca brinda dos interfaces que implementan y solucionan los procesos bloqueantes. (Reactor, s.f.)

Hilos de plataforma / *platform threads*

Un hilo de plataforma es una implementación de un hilo a nivel de sistema operativo (SO). En ese sentido un hilo de plataforma ejecuta código Java en el hilo subyacente al SO y captura a este durante el ciclo de vida del ciclo de ejecución del hilo de plataforma. Como resultado, la cantidad de hilos de plataforma están limitadas por la cantidad de hilos del SO.

Los hilos de plataforma suelen tener una mejor performance en manejo de recursos propiamente mantenidos por el sistema operativos, puesto que esta adheridos a este, así como también permiten el uso de variables locales dentro de cada hilo de ejecución. (Oracle, JAVA SE 20 - Core Libraries, 2023)

Hilos virtuales / *virtual threads*

Un hilo virtual es una instancia de los hilos de plataforma, sin embargo, su ejecución no está vinculada a un hilo específico del sistema operativo (SO). Un hilo ejecuta su tarea en un hilo del sistema operativo a través de una operación bloqueante, una vez concluido el mismo Java suspende este hilo virtual hasta requerir una nueva ejecución y libera el hilo físico, es por lo que este hilo físico puede ser ocupado por otro nuevo hilo virtual para que pueda realizar sus diferentes operaciones. En ese sentido los hilos virtuales van inyectándose en los hilos del SO. (Alarcón, 2023)

Los hilos virtuales se implementan de manera similar a la memoria virtual. Es decir, para simular grandes cantidades de memoria, el sistema operativo asigna recursos de la memoria RAM a fin de que las diferentes operaciones puedan ser cargadas en estas direcciones virtuales. No obstante, es importante indicar que los hilos virtuales al final se acaban ejecutando en los hilos físicos del sistema operativo, sin embargo, se tiene una mejor gestión de estos y permiten la sobrecarga de hilos de manera directa. (Cañas Vargas & Nicholas, 2004)

Respecto al alcance los hilos virtuales tienen poco alcance a nivel de recursos del sistema operativo, puesto que su tiempo de actividad en el hilo del SO es mínimo, por lo que se recomienda su uso para tareas que pueden ser bloqueantes. (Oracle, JAVA SE 21 - Core Libraries, 2023)

Amazon Corretto 21

Es una distribución gratuita, multiplataforma y libre para el uso en producción basada en la especificación OpenJDK administrada y soportada por Amazon. Esta versión tiene la certificación de compatibilidad con el estándar de Java SE ofrecido por Oracle. (Amazon, s.f.)

Máquina virtual - *Droplets*

Para el presente estudio se usará una máquina virtual hospedada en la nube de DigitalOcean, esto con el fin de que la maquina solo cuenta con el software mínimo

necesario para evitar consumos por otras aplicaciones de un computador regular tales como interfaces graficas o procesos en segundo plano asociados al uso de estas interfaces, cualquier tipo de configuración adicional requiere del uso de la consola del sistema operativo elegido (DigitalOcean, DigitalOcean - Droplets, s.f.) , además de no requerir grandes pasos de configuración para las pruebas a realizarse más que instalar el Java JDK 21 o equivalentes.

Material

Hardware asociado:
Región: New York
Datacenter: New York – Datacenter 1 – NYC1

Sistema operativo: Ubuntu 22.04 (LTS) x64

Tipo/capacidad de disco: SSD – 10 GB
CPU: 1 CPU
RAM: 512MB
Software instalado:
Amazon Corretto 21

Métodos

Para el presente estudio usaremos un método basado en un muestreo por selección intencionada o muestreo por conveniencia. Es decir, elegimos a través de métodos no aleatorios la muestra la cual sea representativa de la población objetivo. Este método es útil cuando obtenemos una primera prospección de la población o cuando no existe un marco de la población definida. (Casal & Mateu, 2003)

En ese orden de ideas, se realizarán 100 ejecuciones usando los hilos virtuales y otras 100 usando los hilos físicos, posteriormente se procederán a normalizar estos valores a fin de poder estadísticos claros que puedan representar la mejora en los tiempos de ejecución. Así también el algoritmo a usarse buscará realizar una operación n^2+1 para los 1000 primeros valores enteros positivos.

```
import java.time.Duration;

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.*;

public class Task implements Callable<Integer> {
```


DOI: https://doi.org/10.52248/eb.Vol4Iss3.118

```
    private final int number;

    public Task(int number) {

        this.number = number;

    }

    @Override

    public Integer call() {

        System.out.println("Thread " + Thread.currentThread().threadId() + " - " + number
* number + 1);

        try {
            Thread.sleep(Duration.ofMillis(1000));
        } catch (InterruptedException e) {

            System.out.println("Thread " + Thread.currentThread().threadId() + " - " +
number * number + 1);

            return -1;

        }

        System.out.println("Thread " + Thread.currentThread().threadId() + " - " + number
* number + 1);

        return ThreadLocalRandom.current().nextInt(100);

    }

    public static void main(String[] args) throws InterruptedException,
ExecutionException {

        final ExecutorService executor = Executors.newFixedThreadPool(10);
        List<Task> tasks = new ArrayList<>();
        for (int i = 0; i < 1_000; i++) {
```


DOI: https://doi.org/10.52248/eb.Vol4Iss3.118

```
            tasks.add(new Task(i));

        }

        long time = System.currentTimeMillis();

        executor.invokeAll(tasks);

        time = System.currentTimeMillis() - time;
        System.out.println("time=" + time);
    }

}
```

Figura 1. Código para obtener los tiempos de solución con el uso de hilos físicos.

```
import java.time.Duration;

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.*;

public class Task implements Callable<Integer> {

    private final int number;

    public Task(int number) {

        this.number = number;

    }

    @Override

    public Integer call() {

        System.out.println("Thread " + Thread.currentThread().threadId() + " - " + number
* number + 1);
```

```
        try {
            Thread.sleep(Duration.ofMillis(1000));
        } catch (InterruptedException e) {

            System.out.println("Thread " + Thread.currentThread().threadId() + " - " +
number * number + 1);

            return -1;

        }

        System.out.println("Thread " + Thread.currentThread().threadId() + " - " + number
* number + 1);

        return ThreadLocalRandom.current().nextInt(100);

    }

    public static void main(String[] args) throws InterruptedException,
ExecutionException {

        final ExecutorService executor = Executors.newVirtualThreadPerTaskExecutor();
        List<Task> tasks = new ArrayList<>();

        for (int i = 0; i < 1_000; i++) {

            tasks.add(new Task(i));

        }

        long time = System.currentTimeMillis();

        executor.invokeAll(tasks);
        time = System.currentTimeMillis() - time;
        System.out.println("time=" + time);

    }

}
```

Figura 2. Código para obtener los tiempos de solución con el uso de hilos virtuales

DOI: https://doi.org/10.52248/eb.Vol4Iss3.118

RESULTADOS

Luego de 100 ejecuciones obtenemos tiempos de respuesta medios para el uso de hilos virtuales en 1.071 segundos. Usando hilos de plataforma la misma ejecución llevo un promedio de 100.050 segundos.

Además, el uso de hilos de plataforma o hilos físicos asociados al sistema operativo, presentan la dificultad que no podemos solicitar a java la creación de muchos hilos, puesto que la creación de cada hilo conlleva a que el sistema operativo separe un único hilo para la ejecución, es por ello que en el código correspondiente al uso de hilos de plataforma solo solicitamos la creación de 10 hilos, lo cual puede afectar en el tiempo final de respuesta.

También es importante destacar que los tiempos de respuesta se van estabilizando a través de más ejecuciones, esta es una característica propia de la Java Virtual Machine (JVM), puesto que a más ejecuciones la JVM va optimizando o priorizando las interfaces usadas a fin de mejorar el rendimiento. (GraalVM, s.f.)

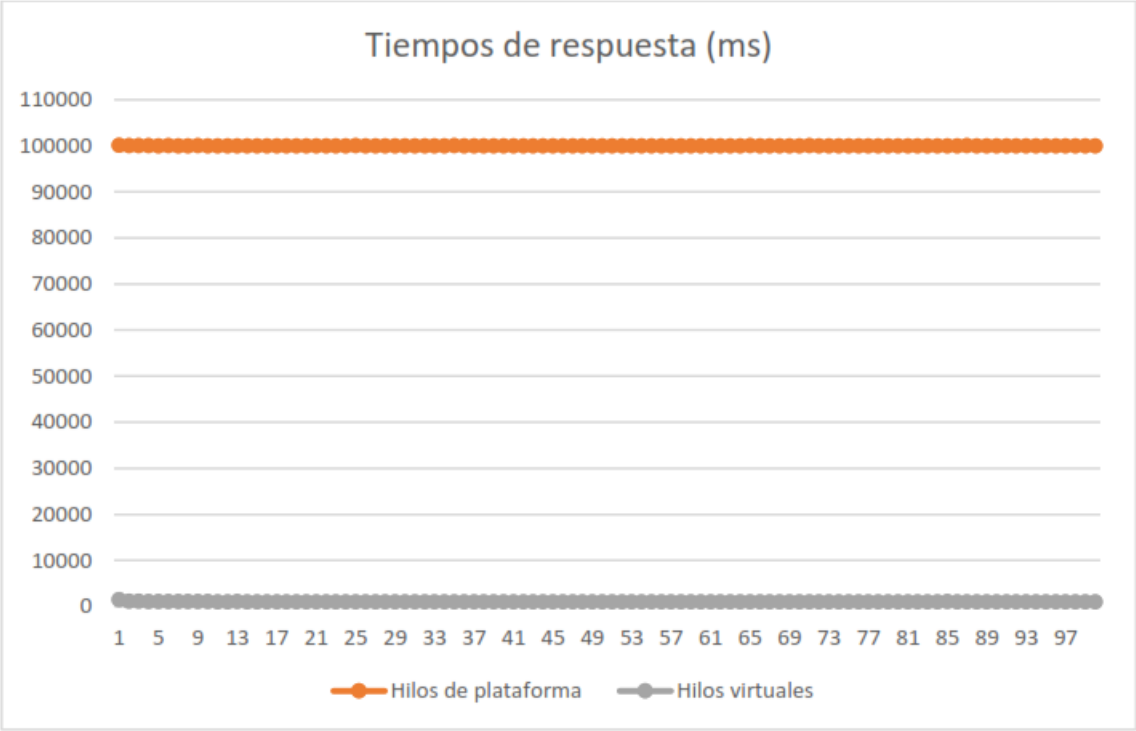

Figura 3. Grafica de tiempos de respuesta para los hilos físicos y los virtuales

Estadisticos	Hilos fisicos	Hilos virtuales
Media	100050.41	1071.76
Mediana	100048	1060.5
Moda	100048	1056
Desviación estandar	11.9173034	50.20344871

Figura 4. Valores estadísticos del os tiempos de respuesta para los hilos físicos o de plataforma y los virtuales

DISCUSIÓN

Podemos concluir que los hilos virtuales conllevan a una mejora en la resolución de diferentes tipos de algoritmos, no obstante, es importante comprender sus limitantes o problemas en su implementación puesto que al ser un nuevo paradigma de programación tiene unos desafíos por resolver, así como *bugs* o problemas que aún se están solucionando en las diferentes actualizaciones a partir de la versión 21 del JDK de Java. Así también hay que considerar que no todos los programas o problemas pueden ser implementados a través de este nuevo paradigma puesto que un requisito fundamental es que las operaciones simultaneas no tengan interconexión o interdependencia, puesto que se pueden generar errores lógicos o de cálculos.

Existen herramientas tales como GraalVM que transpilan el código de Java a un lenguaje binario propio del sistema operativo lo cual mejora los tiempos de despliegue, mejora la seguridad, el tamaño final del archivo es menor, además el uso de recursos es menor al ser desplegado directamente en lenguaje binario. Lo cual puede llevar a una mejora de un factor hasta de 50 veces sobre los tiempos presentados. (Oleg, 2019)

CONCLUSIONES

Para concluir podemos destacar que los tiempos de respuesta pueden alcanzar una mejora hasta 100 veces más rápido que el uso de hilos físicos o hilos de plataforma, en ese sentido se recomienda el uso de los hilos virtuales para operaciones que pueden ser aplicados a través del paradigma de programación paralela. No obstante, es importante considerar que esta nueva interfaz está disponible solo a partir de la versión 21, la cual ha sido desplegada el 19/09/2023, por lo que su implementación en entornos de producción aun es mínima.

La nueva interfaz conlleva a un menor uso de recursos tanto en memoria como en tiempo de procesamiento, además de no bloquear un hilo de ejecución.

El lenguaje de programación Java está en una constante evolución producto de ello esta nueva versión incluye una interfaz para mejorar el rendimiento.

REFERENCIAS BIBLIOGRAFICAS

Alarcón, J. (21 de septiembre de 2023). *Campus MVP*. Obtenido de Hilos virtuales en Java: la revolución del rendimiento en la plataforma Java: https://www.campusmvp.es/recursos/post/hilos-virtuales-en-java-la-revolucion-del-rendimiento-en-la-plataforma-java.aspx

Amazon. (s.f.). *Amazon Corretto 21*. Recuperado el 31 de octubre de 2023, de https://docs.aws.amazon.com/corretto/latest/corretto-21-ug/what-is-corretto-21.html

Cañas Vargas, A., & Nicholas, C. (2004). *Arquitectura de computadores.* Madrid: McGraw Hill Madrid.

Casal, J., & Mateu, C. (2003). Tipos de muestreo. *Rev. Epidem. Med. Prev.*, 1: 3 - 7.

DigitalOcean. (3 de agosto de 22). *Seguridad de subprocesos en Java*. Obtenido de https://www.digitalocean.com/community/tutorials/thread-safety-in-java

DigitalOcean. (s.f.). *DigitalOcean - Droplets*. Recuperado el 31 de octubre de 2023, de https://www.digitalocean.com/products/droplets

GraalVM. (s.f.). *¿Por qué GraalVM?* Recuperado el 31 de octubre de 2023, de https://www.graalvm.org/why-graalvm/

Guru, R. (s.f.). *Iterator*. Recuperado el 31 de octubre de 2023, de https://refactoring.guru/es/design-patterns/iterator

Guru, R. (s.f.). *Observer*. Recuperado el 31 de octubre de 2023, de https://refactoring.guru/es/design-patterns/observer

Java. (s.f.). *Java Downloads*. Recuperado el 31 de Octubre de 2023, de https://www.oracle.com/pe/java/technologies/downloads/

KhanAcademy. (s.f.). *Computación paralela*. Recuperado el 31 de Octubre de 2023, de https://es.khanacademy.org/computing/ap-computer-science-principles/algorithms-101/x2d2f703b37b450a3:parallel-and-distributed-computing/a/parallel-computing

Oleg, S. (28 de Mayo de 2019). *Aplicaciones Java ligeras nativas de la nube*. Obtenido de https://medium.com/graalvm/lightweight-cloud-native-java-applications-35d56bc45673

Oracle. (03 de 2023). *JAVA SE 20 - Core Libraries*. Obtenido de https://docs.oracle.com/en/java/javase/20/core/virtual-threads.html

Oracle. (10 de 2023). *JAVA SE 21 - Core Libraries*. Obtenido de https://docs.oracle.com/en/java/javase/21/core/index.html

Oracle. (s.f.). *Multithreaded Programming Guide*. Recuperado el 31 de Octubre de 2023, de https://docs.oracle.com/cd/E26502_01/html/E35303/compat-14994.html

Ortiz, J. (14 de Julio de 2020). *Teldat*. Obtenido de La computación paralela: alta capacidad de procesamiento: https://www.teldat.com/es/blog/computacion-paralela-capacidad-procesamiento/

Palma, J., Garrido, M., Sánchez, F., & Quesada, A. (2003). *Programación concurrente.* Madrid: Thomson.

ReactiveX. (s.f.). *ReactiveX*. Recuperado el 31 de Octubre de 2023, de https://reactivex.io/

Reactor, P. (s.f.). *Project Reactor*. Recuperado el 31 de Octubre de 2023, de https://projectreactor.io/

Saez, F., Piccoli, F., Printista, M., & Gallard, R. (2003). Paradigmas de programación paralela. *WICC 2003* (págs. 360 - 364). San Luis: Departamento de informatica - Universidad Nacional de San Luis.

Vásquez, L., Valdez, A., Campos, G., Campos, R., & Hernández, R. (2014). *Programación paralela en una técnica de optimización de tiempos de producción de una empresa.* Barranquilla: Facultad de Ingeniería Mecánica y Eléctrica - Universidad Autónoma de Coahuila.