Máster en Cloud Apps

Desarrollo y despliegue de aplicaciones en la nube

Curso académico 2022/2023

Trabajo de Fin de Máster

Título del trabajo

Autor:

Alejandro Quesada Mendo

Tutor:

Micael Gallego Carillo



## Índice

[Índice 2](#_Toc166772265)

[Resumen 3](#_Toc166772266)

[Introducción y objetivos 4](#_Toc166772267)

[Introducción 4](#_Toc166772268)

[Objetivos 5](#_Toc166772269)

[Diseño de aplicación web y sus variaciones. 7](#_Toc166772270)

[Framework 7](#_Toc166772271)

[Base de datos 8](#_Toc166772272)

[Modelos de programación 8](#_Toc166772273)

[Tests de carga 9](#_Toc166772274)

[Análisis de los resultados 10](#_Toc166772275)

[Sin VT vs con VT 10](#_Toc166772276)

[Imperativo vs Reactivo 10](#_Toc166772277)

[Spring Boot vs Quarkus 10](#_Toc166772278)

[Conclusiones y futuros trabajos 11](#_Toc166772279)

[Conclusiones 11](#_Toc166772280)

[Trabajos futuros 11](#_Toc166772281)

[Bibliografía 12](#_Toc166772282)

## Resumen

¿En qué ha consistido el trabajo?  
¿Qué se ha intentado conseguir?  
¿Por qué tiene sentido el trabajo?

## Introducción y objetivos

### Introducción

El pasado septiembre de 2022, se introdujo una nueva herramienta en Java 19 los llamados *Virtual Threads,* o hilos virtuales, como *preview feature*. A mediados de 2023, los Virtual Threads se incluyeron en el lanzamiento de java 21. Los Virtual Threads tienen origen en el Proyecto *Loom*, una iniciativa *open-source* cuyo objetivo es conseguir un modelo de concurrencia ligero y de alto rendimiento para el lenguaje de programación Java.

Los hilos de plataforma, o hilos del sistema operativo, han sido, durante mucho tiempo, la base sobre la que se han construido la gran mayoría de aplicaciones Java. Cada declaración en cada método se ejecuta dentro de un hilo, y como Java es un lenguaje multihilo, múltiples hilos de ejecución suceden al mismo tiempo. Generalmente, las aplicaciones web gestionan múltiples usuarios de manera concurrente, por lo que tiene sentido dedicar un hilo a cada petición o usuario. Este enfoque de asignar un hilo por solicitud es sencillo de comprender, implementar, depurar y analizar, ya que utiliza la unidad de concurrencia de Java, el hilo de plataforma, para representar la unidad de concurrencia de la aplicación. No obstante, este enfoque está limitado por el número de hilos de plataforma del sistema operativo. Al ser un número limitado, las aplicaciones tienen una capacidad limitada de responder peticiones de usuario simultáneas sin que su rendimiento se vea afectado.

A parte de la limitación del número de hilos de plataforma disponibles, el estilo de un hilo por petición presenta otro problema potencial. En muchos casos, las aplicaciones web dependen de operaciones de entrada-salida durante su ejecución, ya sea uso de bases de datos, servicios de mensajería o comunicación con otras aplicaciones, por ejemplo. Las aplicaciones que necesitan de muchas operaciones de entrada-salida bloquean el hilo de plataforma en el que se están ejecutando. De esta forma, el procesador se encuentra ocioso durante largos periodos de tiempo.

De este problema nacieron los modelos de programación concurrente, que buscaban exprimir el uso del hardware al máximo. Para conseguir su objetivo, estos modelos querían limitar los tiempos de espera del procesador durante las operaciones de entrada salida, y dedicar ese tiempo a responder nuevas peticiones. Este modelo en el que una misma petición puede ser ejecutada por distintos hilos durante su ciclo de vida presenta dos grandes problemas. En primer lugar, los cambios de contexto, en los que se guardan los estados de los hilos y se comienzan a ejecutar otros, son operaciones costosas que requieren tiempo de procesamiento no dedicado a responder peticiones de usuario. En segundo lugar, la implementación de estos modelos de programación implica un aumento significativo en la complejidad del código y mayor dificultad a la hora de depurar posibles errores en el código.

Junto con los modelos de programación concurrente, la otra gran solución que surgió para hacer frente a este problema es la programación no bloqueante o reactiva. La programación no bloqueante es un modelo de programación en el que los hilos de plataforma no quedan bloqueados durante las operaciones de entrada/salida. En lugar de esperar a que una tarea se complete antes de pasar a la siguiente, los modelos de programación no bloqueante emplean técnicas como callbacks, promesas o async/await en lenguajes de programación para permitir que el código continúe ejecutándose mientras se espera que ciertas operaciones se completen en segundo plano. Esto puede mejorar significativamente la eficiencia y capacidad de respuesta de las aplicaciones al evitar bloqueos innecesarios y permitir que múltiples operaciones se realicen de manera concurrente. No obstante, también puede introducir desafíos adicionales en la gestión de la concurrencia y la sincronización de datos

Los hilos virtuales, nacen con la intención de resolver el problema de la limitación en el número de hilos de plataforma disponibles, sin renunciar a la comodidad del estilo de un hilo por petición. Como las aplicaciones Java no se ejecutan directamente sobre el sistema operativo, sino sobre la Máquina Virtual de Java, o JVM, Java ha podido romper la relación 1:1 que existía entre los hilos de ejecución y los hilos de plataforma. Se puede decir que un hilo virtual es una instancia de java.lang.Thread que no está atado a un hilo de plataforma en concreto.

De esta forma, se puede mantener el estilo de un hilo por petición usando hilos virtuales. Sin embargo, los hilos virtuales solo consumen recursos de los hilos de plataforma en los momentos en los que haya que realizar operaciones que usen el procesador. Cuando el código que se ejecuta en un hilo virtual hace una operación de entrada/salida bloqueante (usando la API de Java), automáticamente se suspende la ejecución de hilo virtual hasta que se pueda reanudar más adelante. De cara a los desarrolladores, los hilos virtuales son hilos baratos de crear y virtualmente infinitos.

Por eso, los hilos virtuales parecen ser una revolución en el mundo del desarrollo de aplicaciones Java. Se presentan como una solución idílica al problema de la limitación de recursos de los procesadores, prometiendo la misma eficiencia que los modelos de programación concurrente, si no más, ya que no existe la necesidad de hacer costosas operaciones de cambio de contexto. A su vez, sin el aumento en complejidad que traían estos modelos. Y, por último, de manera transparente al usuario, ya que la gestión de los hilos virtuales sucede internamente en la JVM.

### Objetivos

El objetivo principal de este proyecto de fin de máster es, por un lado, analizar el impacto en rendimiento del uso de los hilos virtuales en la ejecución de aplicaciones web Java y, por otro lado, entender la facilidad de adaptación de aplicaciones ya existentes al uso de esta tecnología.

Más específicamente, se quiere poner a prueba las diferencias en rendimiento del uso de hilos virtuales en distintas variaciones de una misma aplicación web con acceso a una base de datos.

Se van a comparar 14 versiones distintas de la misma aplicación, usando las siguientes cuatro variables:

* Uso y no uso de hilos virtuales durante la ejecución.
* Framework de desarrollo: Spring Boot y Quarkus.
* Modelo de programación bloqueante y no bloqueante.
* Base de datos relacional y no relacional.
* Uso y no uso de hilos virtuales durante la ejecución.
* Framework de desarrollo: Spring Boot, como estándar en la industria y Quarkus, como un proyecto mucho más joven y adaptado a las tendencias actuales. Pese a que la experiencia del desarrollador es bastante similar con ambos frameworks, su funcionamiento interno puede estar más o menos optimizado para la ejecución sobre hilos virtuales.
* Modelos de programación bloqueante y no bloqueante, solo en el caso de Spring Boot. Aunque parezca poco intuitivo, Spring Boot permite la ejecución de código no bloqueante sobre hilos virtuales. Quarkus, sin embargo, no lo permite.
* Bases de datos: Mysql, como ejemplo de sistema de gestión de bases de datos relacional, y MongoDB, como ejemplo de sistema de gestión de bases de datos no relacional. Las librerías de comunicación con estos dos tipos de bases de datos pueden estar optimizadas para su ejecución sobre hilos virtuales o no, tanto en Spring Boot como en Quarkus.

## Diseño de aplicación web y sus variaciones.

Durante este apartado se va a introducir la aplicación web Java que se ha usado para realizar las comparaciones de rendimiento y cuáles son sus 14 variaciones.

La aplicación web consiste en un sistema de gestión de películas. La aplicación expone una API que permite hacer las siguientes operaciones:

* Obtener una película por su identificador.
* Obtener una lista paginada de 20 películas.
* Guardar una nueva película en el sistema.
* Actualizar los datos de una película.
* Actualizar la nota de una película.
* Borrar una película del sistema.

La aplicación se comunica con una base de datos, donde se almacenan las películas.

Se han desarrollado catorce versiones de la aplicación, de las cuales, ocho se ejecutan sobre hilos de plataforma y seis sobre hilos virtuales.

A continuación, se detallan las variaciones de la aplicación.

### Framework

Se ha implementado la aplicación usando dos de los frameworks más usados para el desarrollo de aplicaciones web Java.

Por un lado, Spring Boot es, actualmente, el framework de referencia para el desarrollo de aplicaciones web Java en la industria. Es un framework con una larga trayectoria y una gran comunidad de usuarios.

Por otra parte, Quarkus es un framework relativamente joven, concebido para adaptarse a las nuevas corrientes de desarrollo y orientado a la nube y a su ejecución en entornos kubernetes. Tiene menos usuarios y trayectoria que Spring Boot.

Aunque la experiencia de uso de ambos frameworks desde el punto de vista del desarrollados es bastante similar, su funcionamiento interno, librerías propias y gestión de recursos es bastante diferente, como se puede comprobar en la sección de análisis de los resultados.

He creído interesante comparar el rendimiento de los hilos virtuales en dos frameworks distintos para poder discernir si el potencial aumento en el rendimiento y eficiencia solo tiene que ver con la actualización de la JVM o no.

### Base de datos

Se han usado dos sistemas de gestión de bases de datos distintos, uno relacional, Mysql, y otro no relacional, MongoDB.

Se han usado estos dos sistemas por ser sistemas populares frecuentemente usados en el desarrollo de aplicaciones web y por gestionar los datos de manera distinta internamente. Cualquier dupla de sistemas relacional y no relacional hubiese servido, aunque los resultados no necesariamente hubiesen sido los mismos.

### Modelos de programación

La implementación de la aplicación se ha realizado usando dos paradigmas de programación diferentes.

En primer lugar, el modelo de programación bloqueante es aquel que permite que una operación o tarea impide que otras operaciones se ejecuten hasta que se complete. En contraposición, el modelo de programación no bloqueante se refiere a un enfoque de programación que permite que múltiples tareas o procesos se ejecuten de manera concurrente sin bloquear el hilo principal de ejecución.

Para la implementación de las versiones no bloqueantes de la aplicación se ha usado la librería Spring Reactor en Spring Boot y Mutiny en Quarkus.

Spring Boot permite la ejecución de código no bloqueante sobre hilos virtuales. Quarkus no.

## Tests de carga

¿Qué tecnologías se han usado para el desarrollo de los test de carga?

¿En que consistían los test de carga y por qué?

¿Sobre que infraestructura se han ejecutado los test de carga? Docker limitado a 2 cores y 500MB

## Análisis de los resultados

### Sin VT vs con VT

### Imperativo vs Reactivo

### Spring Boot vs Quarkus

## Conclusiones y futuros trabajos

### Conclusiones

¿En qué casos tienen sentido (y en cuáles no) el uso de los VT?

¿Por qué a veces el uso de los VT empeora el rendimiento?

¿Cuál es la mejor combinación de framework-arquitectura de bd-modelo de programación y uso de vt?

¿Los vt merecen la pena para lo que cuesta implementarlos?

### Trabajos futuros

Implementación de otros endpoints en el que la app haga de proxy y se bloquee de manera artificial para comprobar el rendimiento de los VT en ese caso.  
Uso de otras bases de datos/frameworks

Comparar uso de VT contra modelos de programación concurrente “clásicos”

## Bibliografía

### 