Máster en Cloud Apps:

Desarrollo y despliegue de aplicaciones en la nube

Curso académico 2022/2023

Trabajo de Fin de Máster

Análisis y comparativa del rendimiento de aplicaciones web ejecutadas sobre hilos virtuales

Autor: Alejandro Quesada Mendo

Tutor: Micael Gallego Carillo

Índice

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 hilos virtuales tienen origen en el Proyecto *Loom*, una iniciativa código abierto 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.

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.

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 su distinta gestión interna de los datos. Cualquier dupla de sistemas de gestión de bases de datos 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.

Prueba de carga

Para conseguir una comparativa objetiva entre las catorce versiones de la aplicación web, se ha ejecutado la misma prueba de carga a todas las versiones, en un entorno de ejecución similar.

Herramientas utilizadas para el desarrollo de las pruebas de carga

Como el objetivo del proyecto es analizar las diferencias en rendimientos de las distintas versiones de la aplicación, se ha considerado interesante analizar el rendimiento desde dos puntos de vista. En primer lugar, cómo se gestionan los recursos de la máquina en la que se ejecuta la aplicación web. Y, por otro lado, el rendimiento de la aplicación en sí. Es decir, datos relacionados con los tiempos de respuesta y posibles errores de la aplicación, que experimentaría un usuario de esta.

A continuación, se enumeran y explican brevemente las herramientas utilizadas para el desarrollo de las pruebas de carga.

Para el análisis del rendimiento se han usado dos herramientas de código abierto para la monitorización y observabilidad de aplicaciones, Graphana y Prometheus.

Graphana es una plataforma de visualización y análisis de datos que permite crear y compartir dashboards interactivos que muestran datos en tiempo real. Se ha usado Graphana para la obtención de métricas de rendimiento de la memoria y del procesador y del estado de los hilos de plataforma.

Prometheus es un sistema de monitoreo diseñado para la recolección de métricas y su almacenamiento a largo plazo. Prometheus es el encargado de recoger los datos del estado de la aplicación y del sistema en el que se ejecuta y comunicárselos a Graphana para que los muestre en forma de gráfica.

Por otro lado, para el análisis del comportamiento de la aplicación desde el punto de vista del usuario, se ha usado Artillery, una herramienta que permite la realización de pruebas de carga y rendimiento de aplicaciones web. Es especialmente útil para entender cómo se comporta una aplicación bajo condiciones de alta demanda. Artillery permite definir pruebas de carga usando un fichero de configuración, por lo que se ha podido adaptar al caso particular de este trabajo.

Infraestructura usada para las pruebas de carga

Se ha intentado utilizar un entorno aislado durante la ejecución de las pruebas. Para ello, se ha usado Docker para aislar la aplicación del resto del sistema.

El entorno de ejecución de las pruebas tenía a su disposición dos cores del procesador. Por otra parte, la memoria disponible de la Máquina Virtual de Java sobre la que se han ejecutado las distintas versiones de la aplicación durante la ejecución de las pruebas se ha limitado a 500MB.

El diagrama de despliegue que se ha usado para la ejecución de las pruebas de carga es el siguiente:

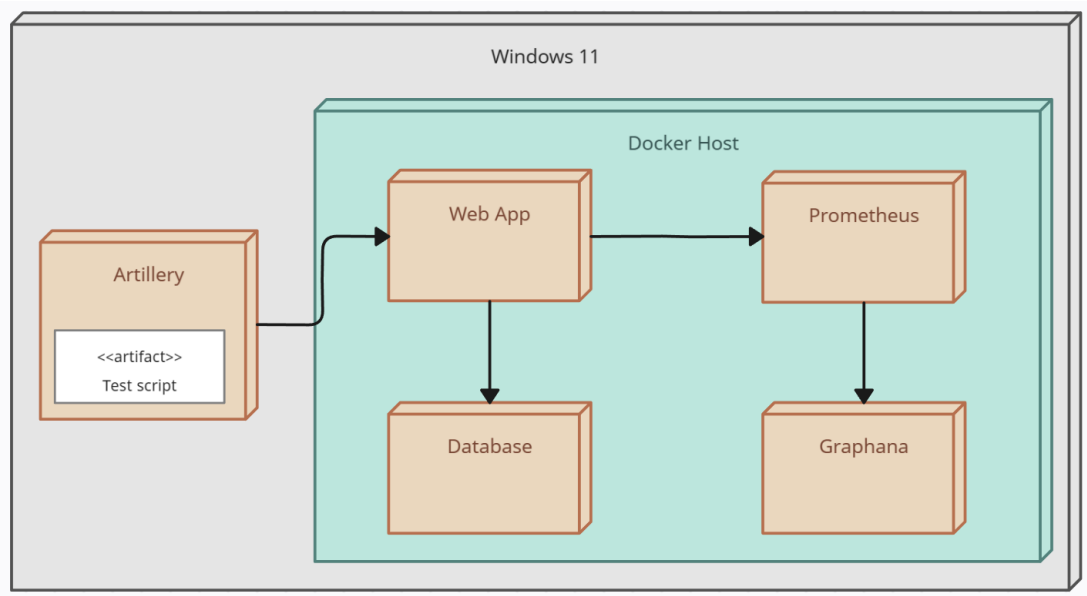

Figura 1: Diagrama de despliegue para la ejecución de pruebas de carga

Ejecución de las pruebas de carga

Para obtener un resultado fiel del rendimiento de las aplicaciones web, se han usado cuatro endpoints de la API que exponen. De cada 100 interacciones con la aplicación, Artillery ejecuta:

- 70 llamadas para obtener una película según identificador.
- 20 llamadas para obtener una página de 20 películas.
- 9 llamadas para guardar una nueva película.
- 1 llamada para eliminar una película según su identificador.

El funcionamiento de la prueba de carga consiste en la realización de llamadas progresivas a la aplicación durante cuatro minutos, de la siguiente forma:

- 200 llamadas por segundo durante 30 segundos
- Aumento a 300 llamadas por segundo durante 30 segundos
- 300 llamadas por segundo durante 30 segundos
- Aumento a 400 llamadas por segundo durante 30 segundos
- 400 llamadas por segundo durante 30 segundos
- Aumento a 500 llamadas por segundo durante 30 segundos
- 500 llamadas por segundo durante 60 segundos

Análisis de los resultados

Durante este apartado se van a analizar los resultados de las pruebas de carga realizadas sobre las diferentes versiones de la aplicación. Como el objetivo de este trabajo es entender los potenciales beneficios de la ejecución de aplicaciones sobre hilos virtuales, el análisis de los resultados se ha realizado comparando los resultados de las versiones equivales con y sin el uso de hilos virtuales durante la ejecución.

Se han analizado los siguientes datos respecto al rendimiento desde el punto de vista del usuario: el tiempo mediano de respuesta, el tiempo medio del percentil 95 y del percentil 99, la ratio peticiones por segundo y las peticiones fallidas por timeouts.

En cuanto a la gestión de recursos, se han seleccionado los siguientes datos para su análisis: porcentaje de uso del procesador, porcentaje de uso del heap de la JVM y uso de los hilos del sistema operativo.

Spring Boot, base de datos relacional y modelo bloqueante.

Los resultados respecto a los tiempos de respuesta se indican en la Figura 2:

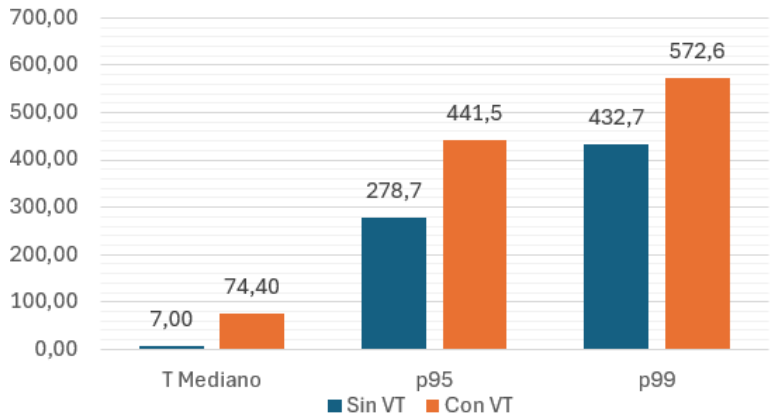

Figura 2: Gráfico comparativo de tiempos de ejecución en milisegundos

La ratio máxima de respuestas por segundo que se alcanzó sin el uso de hilos virtuales fue de 430 peticiones por segundo, mientras que con hilos virtuales fue de 330.

Las peticiones fallidas fueron altas en ambos casos, como se aprecia en la Figura 3.

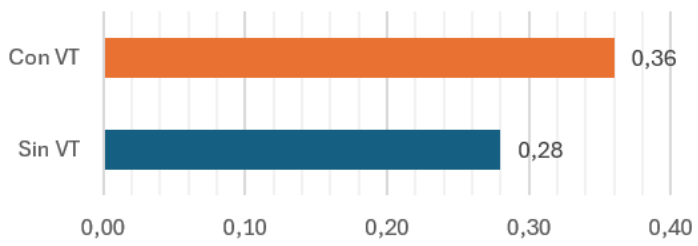

Figura 3: Porcentaje de peticiones fallidas

En cuanto a gestión de recursos, el heap de la JVM tuvo una carga similar en ambas ejecucaciones. Por otro lado el uso de la CPU fue alto en ambos casos como se aprecia en las Figuras 4 y 5:

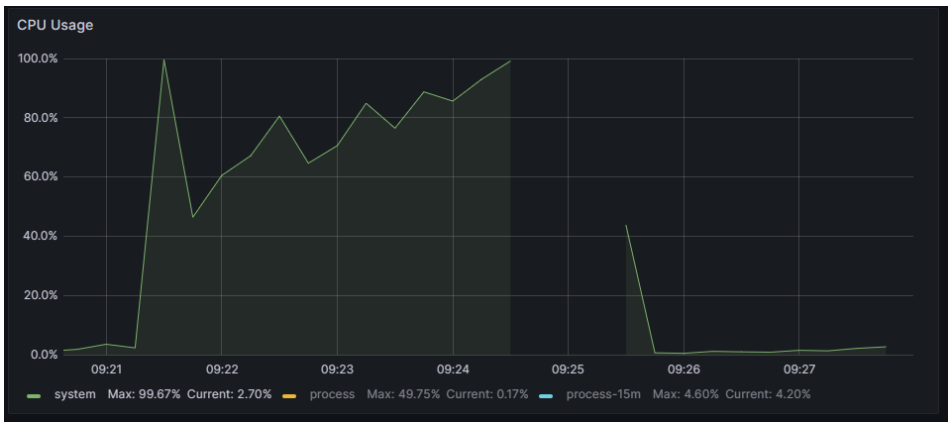

Figura 4: Uso de la CPU aplicación sin uso de hilos virtuales

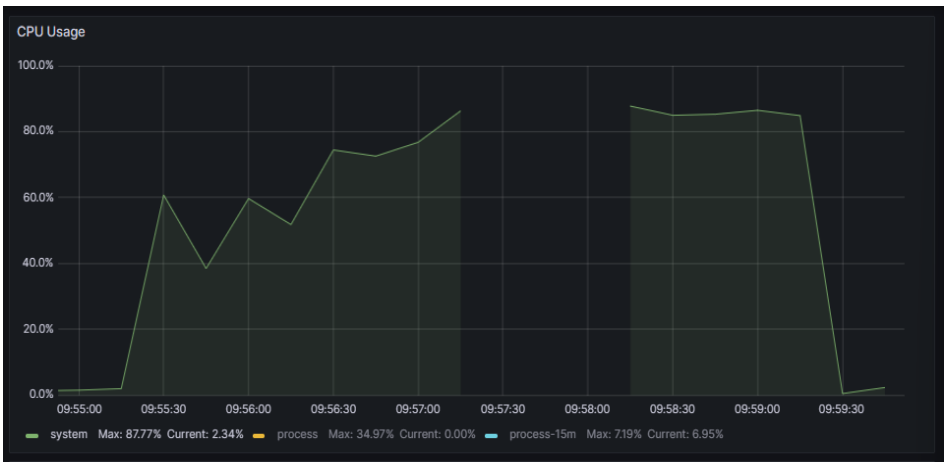

Figura 5: Uso de la CPU aplicación con uso de hilos virtuales

En ambas ejecuciones hubo un pequeño periodo de tiempo en el que la aplicación tuvo tanta carga de trabajo que no fue capaz de mandar las métricas de rendimiento a Graphana. En el caso de ejecución sin uso de hilos virtuales el procesador llego al 100% de uso, mientras que usando hilos virtuales el uso máximo fue del 87,77%.

Por último, en cuanto a los hilos del procesador, durante la ejecución sobre hilos de plataforma, llego a haber 206 hilos bloqueados como se aprecia en la Figura 6. Mientras que, usando hilos virtuales, tan solo hubo 7.

Por lo tanto, se concluye que, pese a una pequeña mejoría respecto a la gestión del procesador, el uso de hilos virtuales para aplicaciones que usen Spring Boot, base de datos Mysql y modelo de programación bloqueante empeora el rendimiento.

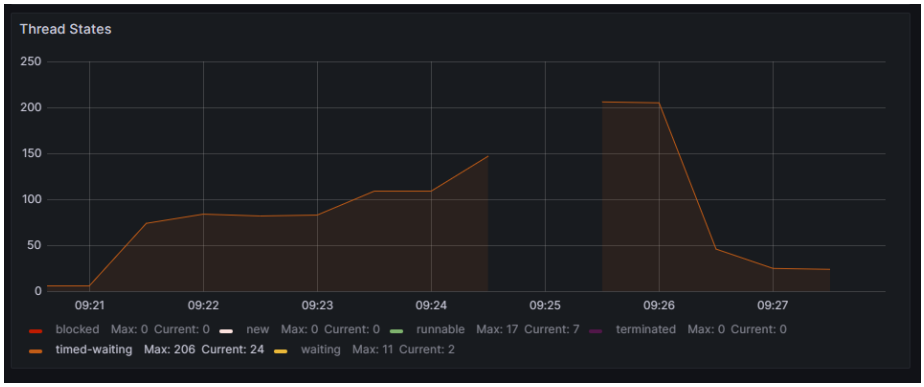

Figura 6: Gestión de hilos durante la ejecución sobre hilos de plataforma

Este resultado puede parecer contraintuitivo ya que la teoría nos dice que los hilos virtuales deberían mejorar el rendimiento. Cabe la posibilidad de que alguna librería usada para la comunicación con la base de datos Mysql, no esté optimizada o bien implementada para su ejecución sobre hilos virtuales, lo que produce la bajada de rendimiento.

Spring Boot, base de datos relacional y modelo no bloqueante

En la Figura 7 se muestran los tiempos de respuesta de ambas versiones. En este caso son similares, salvo por el percentil 99, que es mayor cuando no se usan hilos virtuales.

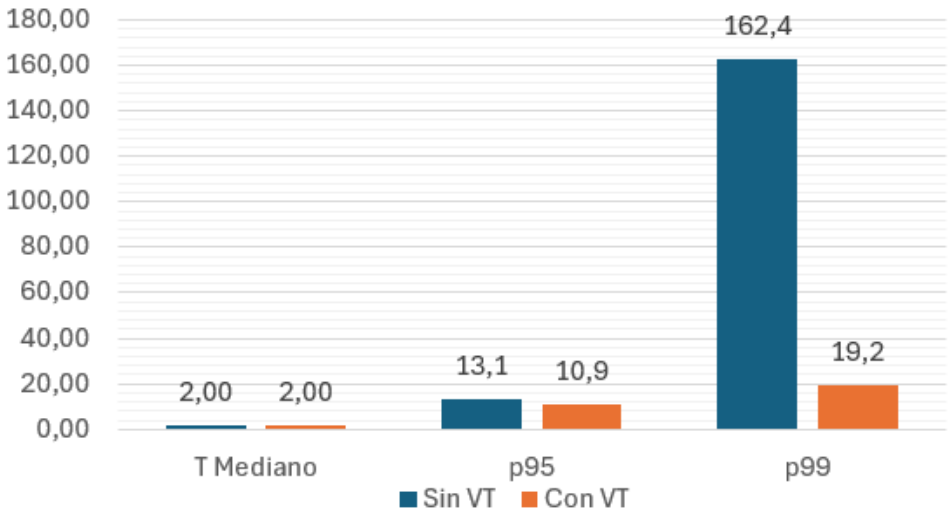

Figura 7: Gráfico comparativo de tiempos de ejecución en milisegundos

En ambos casos, la ratio de peticiones por segundo llego al máximo de 500 y no hubo peticiones fallidas.

En cuanto al uso de la CPU, es similar en ambos casos pese a un pequeño pico del 77% en el caso de uso de los hilos virtuales. Se muestran las gráficas de uso del procesador en las figuras 8 y 9.

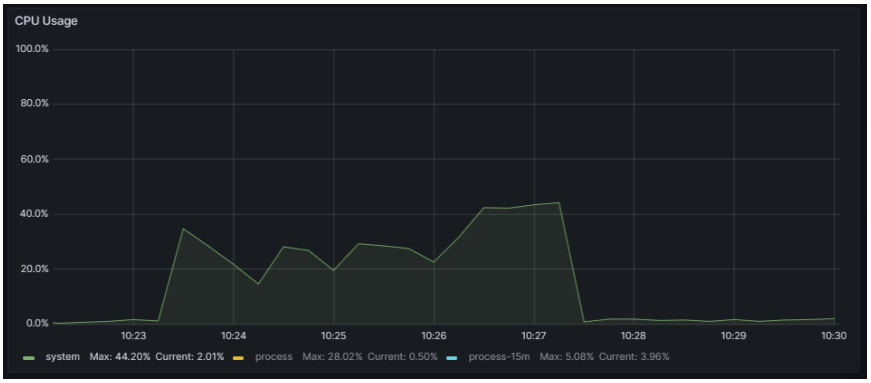

Figura 8: Uso de la CPU aplicación sin uso de hilos virtuales

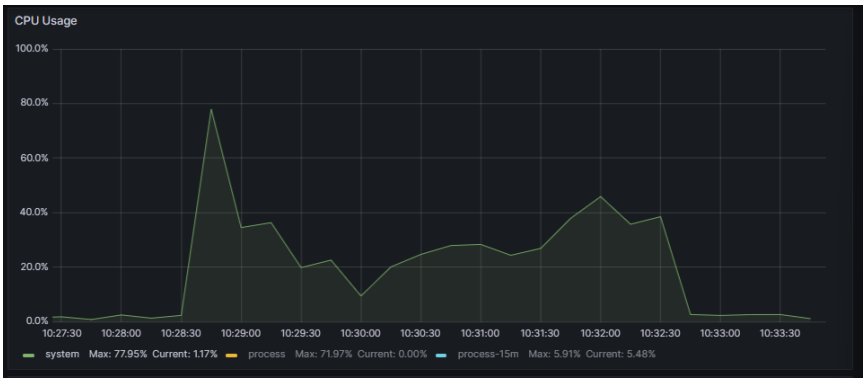

Figura 9: Uso de la CPU aplicación con uso de hilos virtuales

El porcentaje de uso máximo de la memoria de la JVM durante la ejecución sin hilos virtuales fue del 68%, mientras que usando hilos virtuales fue del 52,6%.

Tras analizar los datos, parece que el rendimiento en las dos versiones es similar. Tiene sentido ya que, incluso en el caso de ejecución sin hilos virtuales, se está usando una estrategia que evita el bloqueo de los hilos de plataforma. Por lo que los hilos virtuales tampoco llegan a tener sentido en este caso. De hecho, en ambos casos la aplicación solo llego a tener cuatro hilos bloqueados como máximo de manera constante durante la ejecución. Se muestra el estado de los hilos durante ambas ejecuciones en la Figura 10.

Figura 10: Gestión de hilos durante la ejecución sobre hilos de plataforma

No obstante, sí que mejoran la gestión de la memoria de la máquina virtual de java en casi un 14%.

Spring Boot, base de datos no relacional y modelo bloqueante

En este escenario en concreto, en el que se ha usado MongoDB como base de datos, los resultados son muy similares. En la Figura 11 se muestran los datos relativos a los tiempos de respuesta.

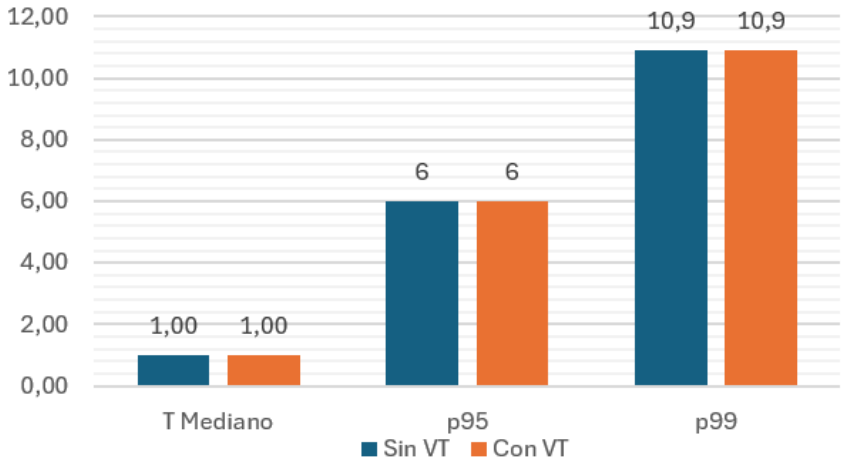

Figura 11: Gráfico comparativo de tiempos de ejecución en milisegundos

En cuanto a la gestión de recursos de ambas versiones, tanto el uso del procesador como el uso de la memoria de la JVM son similares.

La mayor diferencia se encuentra en la gestión de los hilos del sistema operativo, ya que al usarse un modelo de programación bloqueante y al haber una alta carga de trabajo, la versión que no usa hilos virtuales debería llegar a bloquear muchos mas hilos. La realidad es que esta versión de la aplicación es tan eficiente que apenas hay diferencia.

Las Figuras 12 y 13 muestran la gestión de los hilos del sistema operativo en las dos versiones de la aplicación. En el caso de la ejecución sobre hilos de plataforma se llegan a bloquear 21 hilos, mientras que, usando hilos virtuales, 8. Una diferencia mucho menos que en el primer caso analizado en el que la diferencia era de 206 a 7.

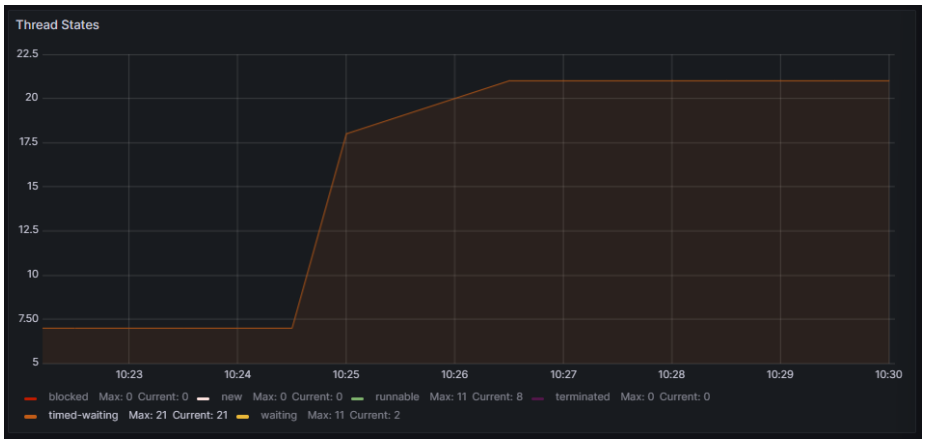

Figura 12: Gestión de hilos durante la ejecución sobre hilos de plataforma

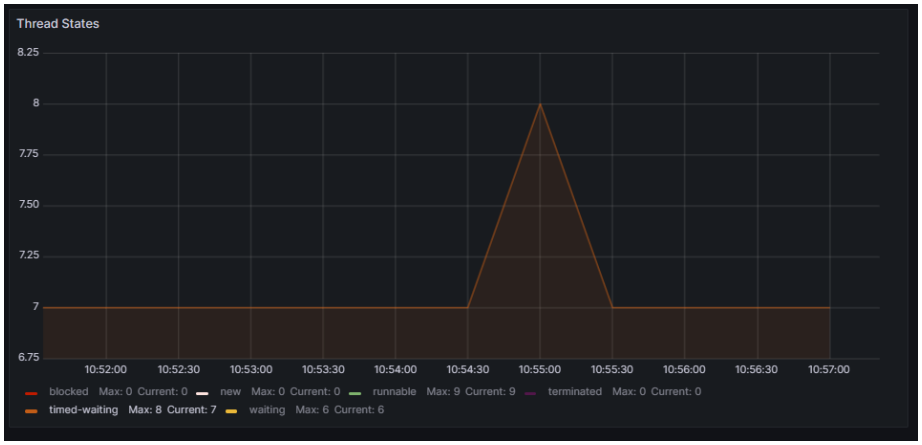

Figura 13: Gestión de hilos durante la ejecución sobre hilos virtuales

Con los datos obtenidos no queda claro cual de las dos versiones es mas eficiente. En ambos casos el rendimiento es similar y superior a la versión equivalente que usa la base de datos relacional.

Spring Boot, base de datos no relacional y modelo no bloqueante

Para el caso de Spring Boot usando un modelo de programación no bloqueante y MongoDB como base da datos, de nuevo los resultados son bastante similares si comparamos la ejecución sobre hilos virtuales y sobre hilos de plataforma.

Los tiempos de respuesta de ambas versiones son prácticamente idénticos. Se muestran los datos en la Figura 14.

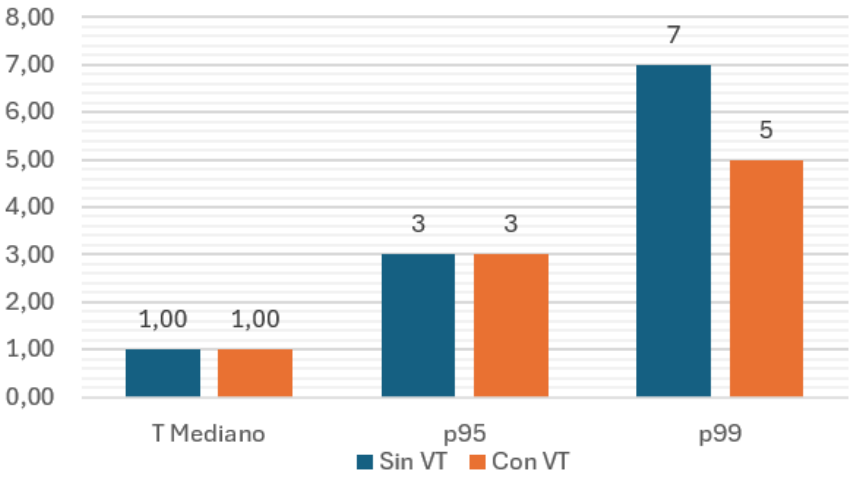

Figura 14: Gráfico comparativo de tiempos de ejecución en milisegundos

Sobre la eficiencia en recursos, la versión que usa hilos virtuales mejora levemente la eficiencia de uso del procesador (Figuras 15 y 16), mientras que, el porcentaje de uso de la memoria de la JVM y la gestión de los hilos del sistema son muy parecidas.

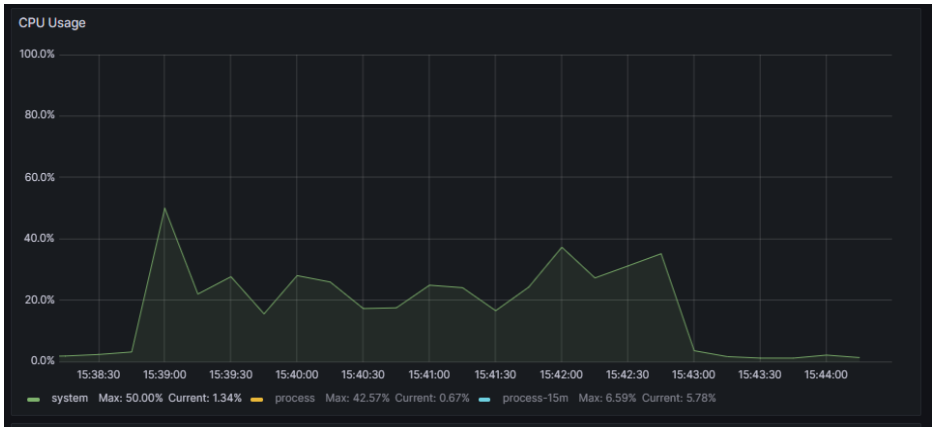

Figura 15: Porcentaje de uso de la CPU aplicación sin uso de hilos virtuales

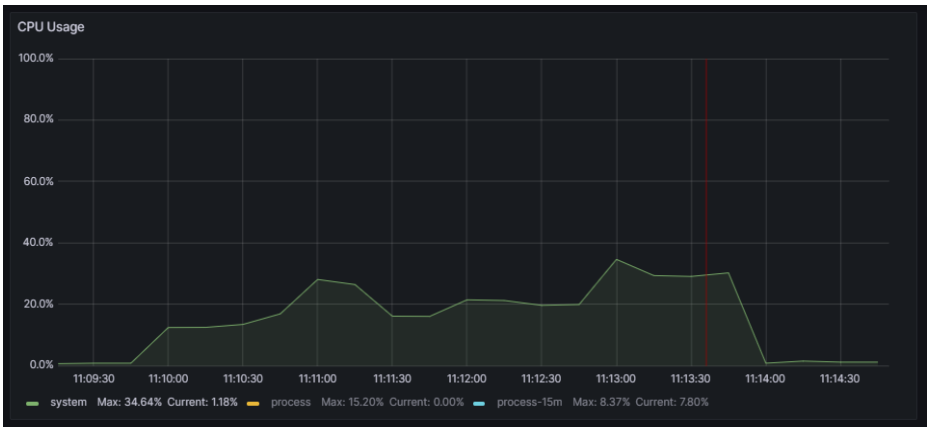

Figura 16: Porcentaje de uso de la CPU aplicación con uso de hilos virtuales

Se puede concluir que el rendimiento es algo mejor cuando la ejecución se realiza usando hilos virtuales, pero la mejora es prácticamente inapreciable desde el punto de vista del usuario.

Quarkus, base de datos relacional y modelo bloqueante

En la Figura 17 se muestran los tiempos de respuesta de las dos versiones implementadas usando Quarkus, Mysql y un modelo de programación bloqueante.

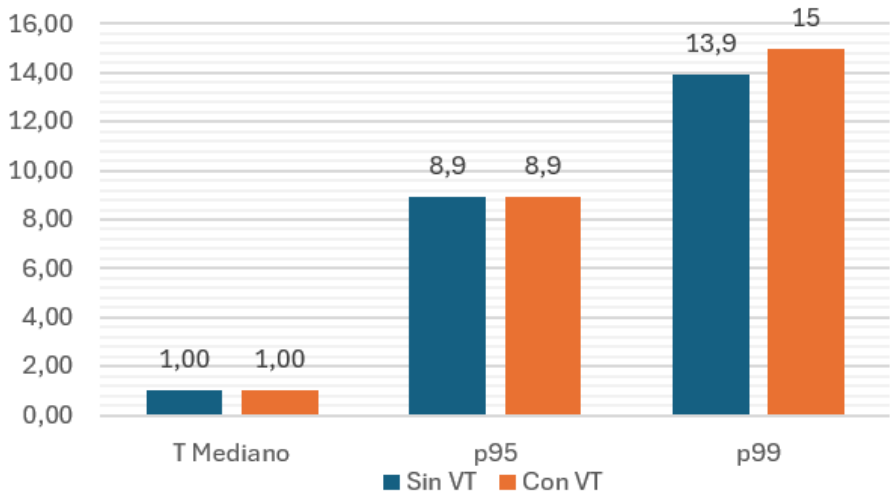

Figura 17:Gráfico comparativo de tiempos de ejecución en milisegundos

Mientras que los tiempos de respuesta son muy similares entre sí, la gestión de recursos si que muestra diferencias significativas. El porcentaje de uso del procesador es menor y más equilibrado durante la ejecución de la prueba de carga en el caso de uso de hilos virtuales, siempre por debajo del 30% de uso. El uso del procesador durante la ejecución sobre hilos de plataforma, aunque es bastante eficiente, es algo superior y tuvo un pico del 40% al comenzar la prueba de carga. Las figuras 18 y 19 muestran las diferencias.

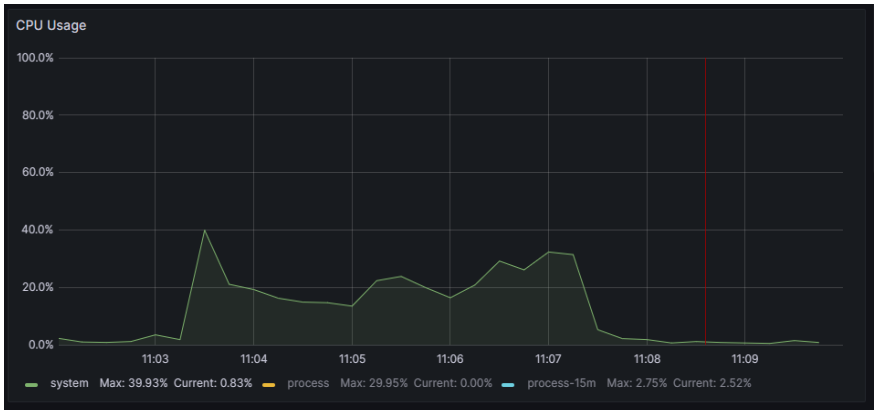

Figura 18: Porcentaje de uso de la CPU aplicación sin uso de hilos virtuales

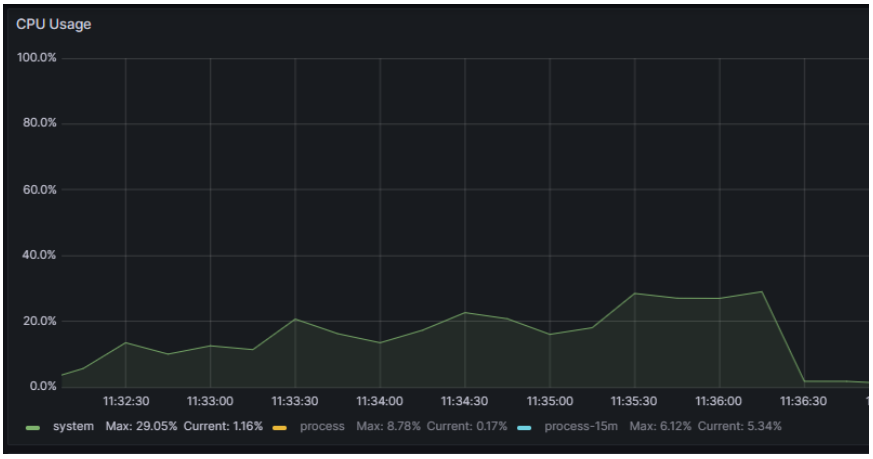

Figura 19: Porcentaje de uso de la CPU aplicación con uso de hilos virtuales

La gestión de la memoria de la JVM es bastante similar y en la gestión de los hilos del sistema, la ejecución sobre hilos virtuales tan solo bloqueó 6 mientras que su contraparte bloqueó 145 en total.

Aunque desde el punto de vista del usuario ambas versiones se compartan de manera parecida, el uso de hilos virtuales mejora la eficiencia de uso de los recursos del sistema.

Quarkus, base de datos no relacional y modelo bloqueante

El último caso que se analiza es el de Quarkus, MongoDB y modelo de programación bloqueante. Este caso es similar al de Spring Boot y Mysql, ya que la combinación de framework y sistema de gestión de base de datos no consigue una gran eficiencia.

La Figura 20 muestra los datos sobre tiempos de respuesta.

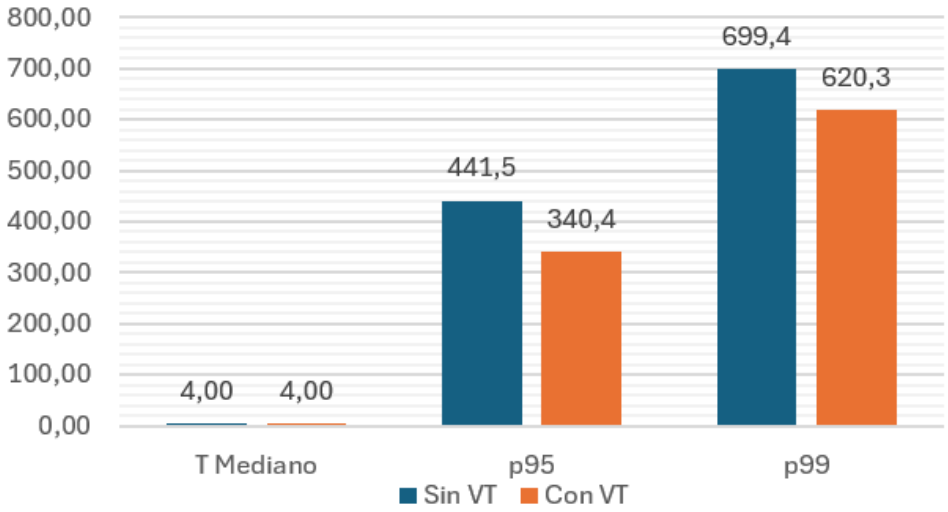

Figura 20: Gráfico comparativo de tiempos de ejecución en milisegundos

En ambos casos, el rendimiento cae notablemente en los percentiles más altos. Llegando a haber respuestas que se demoran mas de medio segundo. Pero los tiempos parecen ser algo mejores durante la ejecución sobre hilos virtuales.

En esta versión sí que ha habido timeouts, 2526 durante la ejecución sin hilos virtuales, frente a 16 cuando se usaron los hilos virtuales. La gráfica de la Figura 21 muestra el porcentaje sobre el total de las peticiones realizadas.

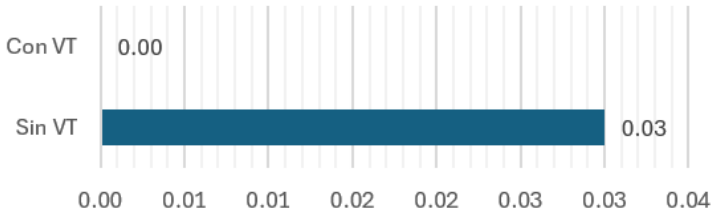

Figura 21:Porcentaje de peticiones fallidas

Por último, la gestión de recursos de las dos versiones es muy similar. En ambos casos, durante los últimos 20 segundos de ejecución de la prueba de carga el procesador alcanzó el 100% de uso. El porcentaje de uso de la memoria fue similar, alcanzando un pico del 32,5% de uso sin hilos virtuales y del 33.8% con hilos virtuales.

Al usarse un modelo bloqueante y al haber llegado al límite de uso del procesador en ambos casos, la gran diferencia está en la gestión de los hilos del sistema, como se aprecia en las Figuras 22 y 23. Durante la ejecución sobre hilos de plataforma llegó a haber 204 hilos bloqueados, mientras que sobre hilos virtuales tan solo hubo 8.

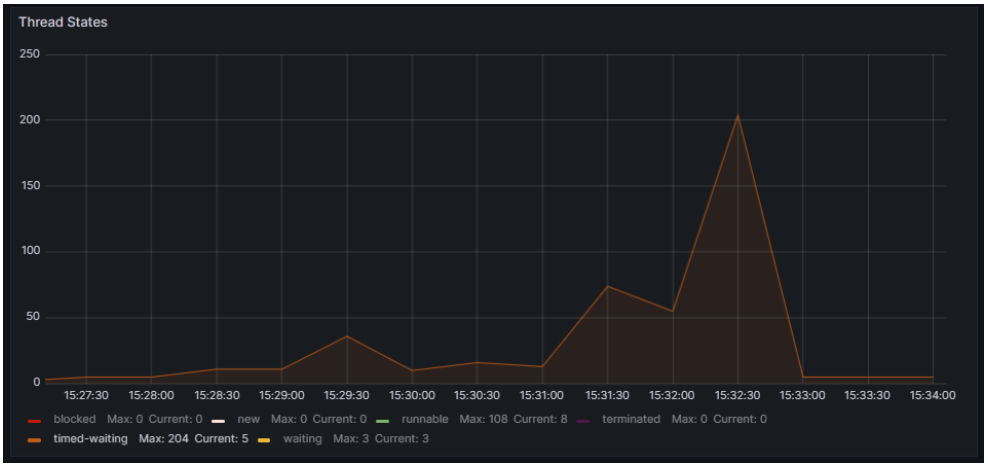

Figura 22: Gestión de hilos durante la ejecución sobre hilos de plataforma

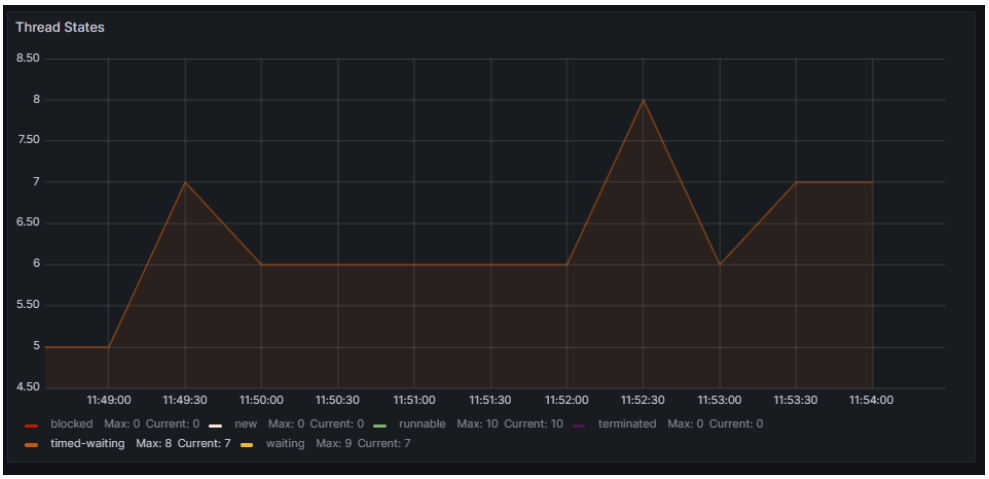

Figura 23: Gestión de hilos durante la ejecución sobre hilos virtuales

Quarkus y modelo no bloqueante

Como se ha mencionado previamente, Quarkus no permite la ejecución de código con modelo no bloqueante sobre hilos virtuales.

Conclusiones y futuros trabajos

Conclusiones

Tras el análisis de los datos resultantes de la ejecución de las pruebas de carga sobre las distintas versiones de la aplicación web, se presentan las conclusiones obtenidas.

Pese a que, en teoría, la llegada de los hilos virtuales debía suponer una revolución en el rendimiento de las aplicaciones Java, en el caso práctico planteado en este proyecto no siempre se mejora el rendimiento, o en caso de mejorarse, la mejoría es sutil.

En las dos comparativas de Quarkus, el rendimiento, sobre todo en la gestión de recursos, mejora cuando la ejecución de las aplicaciones web se realiza sobre hilos virtuales. En el caso de Spring Boot y Mysql, el rendimiento empeoró con el uso de los hilos virtuales. Mientras que el rendimiento de las versiones de Spring Boot y MongoDB es prácticamente idéntico, independientemente del uso de hilos virtuales durante la ejecución.

Al haber obtenido unos resultados tan dispares y variables en función de las tecnologías utilizadas, parece que no hay una respuesta clara a la pregunta sobre cuándo deben usarse los hilos virtuales.

Para un caso de uso más realista, en aplicaciones con mucho mayor flujo de datos y más tecnologías interactuando entre sí, la única forma posible de analizar la potencial mejora en rendimiento mediante el uso de hilos virtuales es hacer pruebas sobre cada aplicación en particular.

Da la sensación de que, al tratarse de una tecnología relativamente novedosa, los softwares de terceros, en los que se apoya el lenguaje Java para la construcción y desarrollo de aplicaciones empresariales, todavía tienen margen de optimización. Probablemente, a medida que el uso de los hilos virtuales pase a ser una práctica más común en la industria, la integración de éstos con las herramientas de terceros también mejore.

En cualquier caso, ya que prácticamente no hay que hacer ningún cambio en el código o en la configuración de una aplicación para que se ejecute sobre hilos virtuales, creo que merece la pena tratar de probar las diferencias en rendimiento, sobre todo en el caso de aplicaciones que requieran de muchas operaciones de entrada/salida durante su ejecución.

Trabajos futuros

De cara a una posible continuación de este proyecto, serían interesantes las siguientes ideas:

- Tratar de poner más al limite la aplicación de prueba y usarla a modo de proxy, llamando a un servicio externo que simule un tiempo de procesamiento determinado. De esta forma los hilos de ejecución podrían estar bloqueados durante más tiempo y posiblemente se apreciaría más el beneficio real del uso de los hilos virtuales.

- Ampliar la comparativa con otros servicios de gestión de bases de datos. Sería interesante saber cuánto del rendimiento de las versiones de la aplicación tendía que ver con el modelo de gestión de los datos de la base de datos y cuanto con el proveedor de ese servicio en sí. De esta forma se podría saber si los hilos virtuales funcionan mejor con un tipo de base de datos que con otro.

- Por último, creo que sería interesante comparar el rendimiento de una aplicación que use hilos virtuales durante la ejecución con el rendimiento de una aplicación que use un modelo de programación concurrente “clásico”, con cambios de contexto y una complejidad mayor del código.

Bibliografía

- Project Loom: https://cr.openjdk.org/~rpressler/loom/Loom-Proposal.html
- Java Virtual Threads Release: https://openjdk.org/jeps/425
- Baeldung: Concurrency vs parallelism: https://www.baeldung.com/cs/concurrency-vs-parallelism
- Spring Boot: Embracing Virtual Threads: https://spring.io/blog/2022/10/11/embracing-virtual-threads
- Baeldung: Working with Virtual Threads in Spring 6: https://www.baeldung.com/spring-6-virtual-threads
- Medium: Boost your applicaction's performance with virtual Threads: Exploring Project Loom: https://medium.com/@knowledge.cafe/spring-boot-virtual-threads-52e28bb0ca5
- Medium: Virtual Threads in Spring Boot with Java 19: https://medium.com/@egorponomarev/virtual-threads-in-spring-boot-with-java-19-ea98e1725058
- Spring 6 presentation: https://www.youtube.com/watch?v=_o7NIaOVjNM
- Spring Boot 3.2.0 release: https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-3.2.0-M1-Release-Notes
- Virtual Threads on Quarkus: https://es.quarkus.io/blog/virtual-thread-1/
- Quarkus: Writing CRUD applications using virtual threads: https://quarkus.io/blog/virtual-threads-2/
- Virtual Threads and Mutiny: https://www.reddit.com/r/quarkus/comments/17nlvls/virtual_threads_and_mutiny/
- Medium: Monitoring Made Simple: Empowering Spring Boot Applications with Prometheus and Grafana: https://medium.com/simform-engineering/revolutionize-monitoring-empowering-spring-boot-applications-with-prometheus-and-grafana-e99c5c7248cf
- Baeldung: How To Configure Java Heap Size Inside a Docker Container: https://www.baeldung.com/java-docker-jvm-heap-size