Usando virtual threads para incrementar dramáticamente la paralelización de tu aplicación

JConf Perú 2023

Andrés Alcarraz

Sábado 2º de Diciembre de 2023

Acerca de mí

Acerca de mí

- Uruguayo 
- 24 años desarrollando en Java.
- @ alcarraz@gmail.com
- linkedin.com/in/andresalcarraz
- X/andresalcarraz
- github.com/alcarraz
- stackoverflow/3444205

Un poco de historia

Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo. [Ora04]

Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo. [Ora04]
- Native threads: Introducidos en java 1.2 (1998)

Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo. [Ora04]
- Native threads: Introducidos en java 1.2 (1998)
- Executors: Introducidos en Java 5.0 (JEP 444 [Jep]).

Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo. [Ora04]
- Native threads: Introducidos en java 1.2 (1998)
- Executors: Introducidos en Java 5.0 (JEP 444 [Jep]).
  - Thread pools: ya no hay que implementarlos a mano o depender de bibliotecas.

Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo. [Ora04]
- Native threads: Introducidos en java 1.2 (1998)
- Executors: Introducidos en Java 5.0 (JEP 444 [Jep]).
  - Thread pools: ya no hay que implementarlos a mano o depender de bibliotecas.
  - Scheduling.

Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo. [Ora04]
- Native threads: Introducidos en java 1.2 (1998)
- Executors: Introducidos en Java 5.0 (JEP 444 [Jep]).
  - Thread pools: ya no hay que implementarlos a mano o depender de bibliotecas.
  - Scheduling.
- Programación reactiva, nombre cool, pero lo cool termina ahí.

Un poco de historia

Historia de la multitarea en Java

- Green threads: Java 1.1. Todos los threads compartían un thread del sistema operativo. [Ora04]
- Native threads: Introducidos en java 1.2 (1998)
- Executors: Introducidos en Java 5.0 (JEP 444 [Jep]).
  - Thread pools: ya no hay que implementarlos a mano o depender de bibliotecas.
  - Scheduling.
- Programación reactiva, nombre cool, pero lo cool termina ahí.
- Java 21: virtual threads, bloquear está bien de nuevo.

Un poco de historia

El problema con los threads nativos

- Costosos de crear.

Un poco de historia

El problema con los threads nativos

- Costosos de crear.
- Costosos de mantener.

Un poco de historia

El problema con los threads nativos

- Costosos de crear.
- Costosos de mantener.
- En llamadas bloqueantes, los recursos quedan bloqueados.

Un poco de historia

El problema con los threads nativos

- Costosos de crear.
- Costosos de mantener.
- En llamadas bloqueantes, los recursos quedan bloqueados.
- Veamos un ejemplo, ¡Coding time!.

Un poco de historia

La solución reactiva

Ejemplo tomado de José Paumard, que a su vez lo toma de Tomasz Nurkiewicz. [Pau23]

```
User user = userService.findUserByName(name);
if (!repo.contains(user)) repo.save(user);

var cart = cartService.loadCartFor(user);
var total = cart.items().stream()
                .mapToInt(Item::price)
                .sum();

var transactionId = paymentService.pay(user, total);
emailService.send(user, cart, transactionId);
```

Un poco de historia

La solución reactiva

```
var future = supplyasync(() -> userService.findUserByName(name))
        .thenCompose(
                user -> allof(
                        supplyAsync(() -> !repo.contains(user))
                        .thenAccept (
                            doesNotContain -> {
                                if (doesNotContain) repo. save(user);
                            }
                        ),
                        supplyAsync(
                            () -> cartservice.loadCartFor(user)
                                .thenapply(
                                    cart ->
                                        supplyAsync(() -> cart.items().stream().mapToInt(Item::price).sum())
                                                .thenapply(
                                                    total -> paymentService.pay(user, total))
                                                    .thenAccept (
                                                        transactionId -> emailService.send(user, cart, transactionId)
                                                    )
                                )
                        )
                )
        )
```

Un poco de historia

La solución reactiva

- Operaciones escritas como lambdas
- El resultado de una es pasado a la siguiente.
- No se debe bloquear en las lambdas.
  - Cuando se está esperando por un recurso externo, retornar con un future.
- Adiós a la programación imperativa.
- La lógica de negocio se mezcla con la lógica del framework $\rightarrow$complica cambios en la lógica de negocio.
- Si una lambda, genera un error en el framework, no es trivial descubrir cual fue.

Propiedades de los Virtual Threads

Propiedades de los Virtual Threads

Ventajas

- Solución nativa.

Propiedades de los Virtual Threads

Ventajas

- Solución nativa.
- Fáciles de usar.

Propiedades de los Virtual Threads

Ventajas

- Solución nativa.
- Fáciles de usar.
- Livianos.

Propiedades de los Virtual Threads

Ventajas

- Solución nativa.
- Fáciles de usar.
- Livianos.
- Código legible.

Propiedades de los Virtual Threads

Ventajas

- Solución nativa.
- Fáciles de usar.
- Livianos.
- Código legible.
- No se necesita hacer pool de ellos.

Propiedades de los Virtual Threads

Ventajas

- Solución nativa.
- Fáciles de usar.
- Livianos.
- Código legible.
- No se necesita hacer pool de ellos.
- Revisitemos el ejemplo anterior.

Propiedades de los Virtual Threads

Desventajas

- No hay una solución nativa para limitar el número de threads.

Propiedades de los Virtual Threads

Desventajas

- No hay una solución nativa para limitar el número de threads.
- Por lo tanto no hay una forma trivial de limitar el acceso a un recurso.

Propiedades de los Virtual Threads

Desventajas

- No hay una solución nativa para limitar el número de threads.
- Por lo tanto no hay una forma trivial de limitar el acceso a un recurso.
- Usar semáforos para controlar esto.

Propiedades de los Virtual Threads

Desventajas

- No hay una solución nativa para limitar el número de threads.
- Por lo tanto no hay una forma trivial de limitar el acceso a un recurso.
- Usar semáforos para controlar esto.
- Se puede crear un executor personalizado para esto.

Propiedades de los Virtual Threads

Como se usan

- Factory method en la clase `Thread`

```
Thread virtual = Thread.ofVirtual().start(
  () -> System.out.println("Hola Mundo!"));
```

- Con executors:

```
ExecutorService service
  = Executors.newVirtualThreadPerTaskExecutor();
...
service.execute(
  () -> System.out.println("Hola Mundo!"));
```

- Por cierto, `ExecutorService` ahora implementa `AutoCloseable`. ¡Se puede utilizar en try-with-resources!

Propiedades de los Virtual Threads

Que hacer

- Bloquear está bien, ¡para eso están!

Propiedades de los Virtual Threads

Que hacer

- Bloquear está bien, ¡para eso están!
- Usarlos para tareas intensivas en el uso de recursos externos.

Propiedades de los Virtual Threads

Que hacer

- Bloquear está bien, ¡para eso están!
- Usarlos para tareas intensivas en el uso de recursos externos.
  - Operaciones en bases de datos.

Propiedades de los Virtual Threads

Que hacer

- Bloquear está bien, ¡para eso están!
- Usarlos para tareas intensivas en el uso de recursos externos.
  - Operaciones en bases de datos.
  - Llamadas a servicios REST.

Propiedades de los Virtual Threads

Que no hacer

- No usarlos para tareas que sólo hagan computación en memoria.

Propiedades de los Virtual Threads

Que no hacer

- No usarlos para tareas que sólo hagan computación en memoria.
- No usarlos para ejecutar parallel streams.

Propiedades de los Virtual Threads

Que no hacer

- No usarlos para tareas que sólo hagan computación en memoria.
- No usarlos para ejecutar parallel streams.
  - No deberíamos ejecutar código bloqueante en parallel streams.

Propiedades de los Virtual Threads

Que no hacer

- No usarlos para tareas que sólo hagan computación en memoria.
- No usarlos para ejecutar parallel streams.
  - No deberíamos ejecutar código bloqueante en parallel streams.
- Evitar fijar el virtual thread a un platform thread por mucho tiempo.

Propiedades de los Virtual Threads

Que no hacer

- No usarlos para tareas que sólo hagan computación en memoria.
- No usarlos para ejecutar parallel streams.
  - No deberíamos ejecutar código bloqueante en parallel streams.
- Evitar fijar el virtual thread a un platform thread por mucho tiempo.
  - No llamar a código nativo que pueda bloquearse.

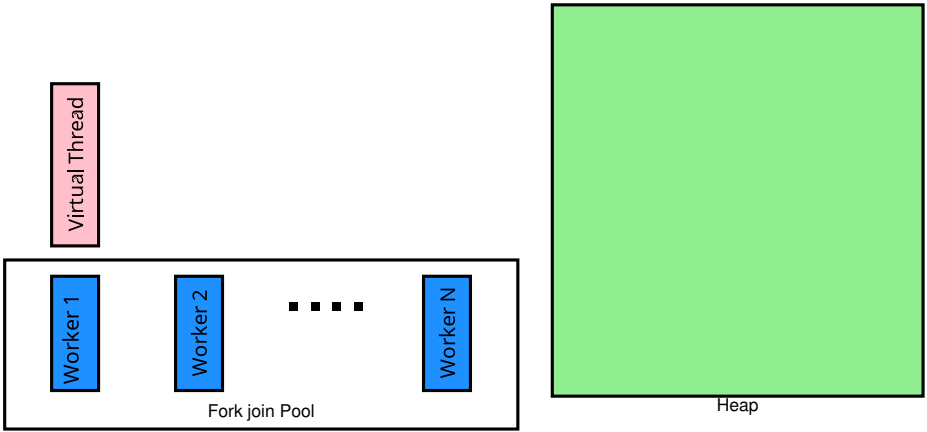
Propiedades de los Virtual Threads

Que no hacer

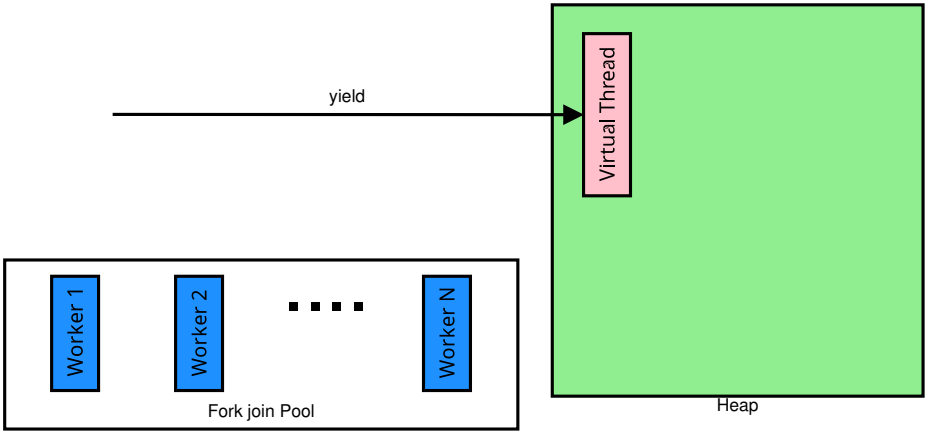
- No usarlos para tareas que sólo hagan computación en memoria.
- No usarlos para ejecutar parallel streams.
  - No deberíamos ejecutar código bloqueante en parallel streams.
- Evitar fijar el virtual thread a un platform thread por mucho tiempo.
  - No llamar a código nativo que pueda bloquearse.
  - No usar bloques `synchronized` para sincronizar cosas que pueden demorar. Usar `ReentrantLock` en su lugar

Detalles de implementación

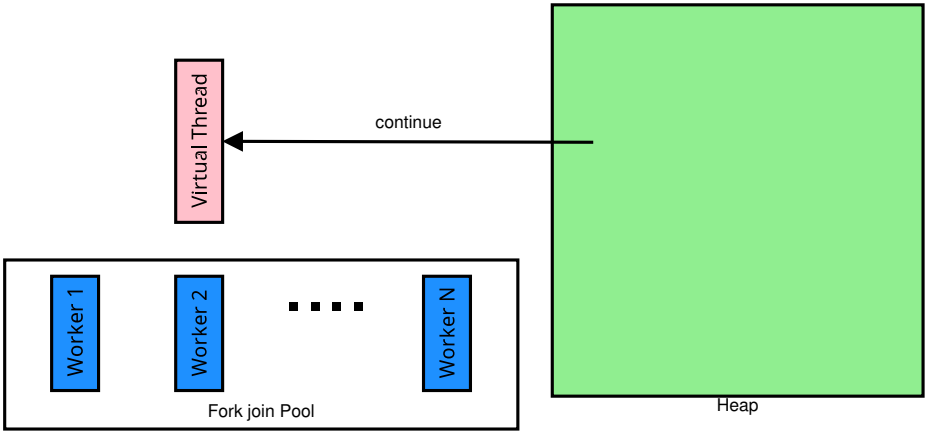
Detalles de implementación



Detalles de implementación



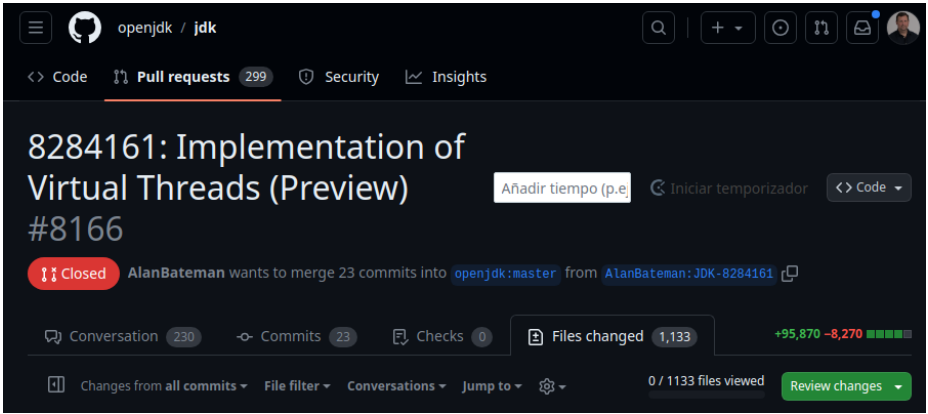
Detalles de implementación



Detalles de implementación

Pull request

Se refactorizó todo el código bloqueante de la JDK, para llamar a `Continuations.yield()`



Detalles de implementación

Thread.*sleep*

```
public static void sleep(long millis) throws InterruptedException {
    if (millis < 0) {
        throw new IllegalArgumentException("timeout value is negative");
    }
    long nanos = MILLISECONDS.toNanos(millis);
    ThreadSleepEvent event = beforeSleep(nanos);
    try {
        if (currentThread() instanceof VirtualThread vthread) {
            vthread.sleepNanos(nanos);
        } else {
            sleep0(nanos);
        }
    } finally {
        afterSleep(event);
    }
}
```

Frameworks

Frameworks

Probablemente no tengas que escribir código para hacer uso de los virtual threads:

- Spring: `spring.threads.virtual.enabled=true`
- Quarkus: `@RunOnVirtualThread`

Otros objetivos de project Loom

Otros objetivos de project Loom

- Structured Concurrency (Preview) JEP 453
- Scoped Values (Preview) JEP 446 (second preview en Java 22: JEP 464)

Ejemplo

Ejemplo

Coding time!

Ejemplo

[Jep] *JEP 444: Virtual Threads*. 2023. URL: https://openjdk.org/jeps/444 (visitado 27-11-2023).

[Ora04] Oracle. *Concurrency Utilities Overview*. 2004. URL: https://docs.oracle.com/javase/1.5.0/docs/guide/concurrency/overview.html (visitado 27-11-2023).

[Pau23] José Paumard. *Java 21 new feature: Virtual Threads, RoadTo21*. 2023. URL: https://youtu.be/5E0LU85EnTI?si=pX3tL808G2AsTOwK (visitado 01-12-2023).