УДК 04.75

Кугаївська С. - ст. гр. КН-421
Відокремлений структурний підрозділ «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»

ВИКОРИСТАННЯ ВІРТУАЛЬНИХ ПОТОКІВ ДЛЯ ОПТИМІЗАЦІЇ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ У ВЕБ-ЗАСТОСУНКАХ

Науковий керівник: викладач-методист, спеціаліст вищої категорії
Марціяш Г.Я.

Kuhaivska S.
Separate Structural Subdivision «Ternopil Professional College of Ternopil Ivan Puluj National Technical University»

USING VIRTUAL THREADS TO OPTIMIZE PARALLEL COMPUTING IN WEB APPLICATIONS

Supervisor: teacher-methodologist, specialist of the highest category
Martsiiyash G.

Ключові слова: паралелізм, віртуальні потоки, високопродуктивні системи.
Keywords: parallelism, virtual threads, high-performance systems.

Сучасна еволюція веб-технологій диктує жорсткі вимоги до продуктивності серверних систем. Зі збільшенням кількості користувачів та переходом до мікросервісної архітектури, здатність системи ефективно обробляти тисячі одночасних з'єднань стала критичним фактором успіху. Протягом останніх десятиліть стандартом для екосистеми Java була модель «один потік на один запит» (thread-per-request). Однак ця модель має фундаментальне обмеження: системні потоки (Platform Threads) є дорогими ресурсами операційної системи. Кожен такий потік резервує близько 1 МБ пам'яті для свого стека, а перемикання між ними на рівні ядра ОС вимагає значних обчислювальних витрат [1].

Якщо говорити про технічну архітектуру Virtual Threads, то віртуальні потоки, впроваджені в межах проєкту Loom (Java 21+), пропонують іншу парадигму. Це реалізація концепції спільнотних потоків (user-mode threads), які повністю керуються середовищем виконання Java (JVM), а не ядром операційної системи [3].

Ключовим технічним аспектом є те, що віртуальний потік не прив'язаний до конкретного системного потоку на весь час свого життя. Замість цього використовується механізм динамічного монтажу. Коли віртуальний потік зустрічає блокуючу операцію (I/O), JVM зберігає його поточний стан у вигляді об'єкта в купі (Heap). Системний потік, на якому виконувався віртуальний потік, звільняється і може бути використаний планувальником (Scheduler) для виконання іншого віртуального потоку. Планувальник у свою чергу ж використовує FIFO-черги та алгоритм крадіжки

завдань (work-stealing) у межах ForkJoinPool, що забезпечує максимальне завантаження всіх ядер процесора без надмірного перемикання контексту ОС [1].

Однією з найважливіших переваг віртуальних потоків є їхня повна інтеграція з існуючим стеком технологій. У Spring Boot 3.2+ підтримка віртуальних потоків дозволяє значно підвищити продуктивність стандартного модуля Spring MVC. Розробники можуть продовжувати писати лінійний, зрозумілий код, використовуючи JDBC або стандартні RestTemplate-запити, але при цьому отримувати масштабованість [2].

На відміну від реактивного програмування, де один логічний запит може виконуватися частинами на різних потоках, що робить стектрейси нечитабельними, віртуальні потоки зберігають цілісність контексту. Інструменти моніторингу та профілювання (наприклад Java Flight Recorder) сприймають віртуальний потік як звичайний потік виконання. Це дозволяє використовувати стандартні механізми ThreadLocal для зберігання контексту безпеки або транзакцій, забезпечуючи детермінованість та легкість діагностики високонавантажених систем [3].

Попри свою інноваційність, віртуальні потоки не є універсальною заміною для всіх видів обчислень. Вони демонструють найкращі результати у задачах, орієнтованих на ввід-вивід (I/O-bound), таких як робота з базами даних (PostgreSQL, MySQL) або HTTP-сервісами. Для задач, що інтенсивно використовують ресурси процесора (CPU-bound), системні потоки залишаються ефективнішими.

Важливим технологічним викликом є явище жорсткої прив’язки віртуального потоку до його системного носія. Така ситуація виникає в моменти, коли код виконується всередині синхронізованих секцій. У ці періоди віртуальний потік втрачає свою гнучкість і блокує фізичний ресурс операційної системи, оскільки середовище виконання не здатне вчасно призупинити його роботу та звільнити потік-носій для виконання інших актуальних завдань. За умов масового виникнення таких затримок ефективність динамічного розподілу навантаження значно знижується, що може нівелювати переваги масштабування.

Висновки. Віртуальні потоки є найбільш значущим оновленням парадигми паралелізму в Java за останні два десятиліття. Вони дозволяють вирішити проблему масштабованості веб-застосунків без переходу до складних асинхронних моделей програмування. Для сучасної програмної інженерії це означає можливість розробки високонавантажених систем із меншими інфраструктурними витратами та вищою якістю коду.

Список використаних джерел

1. A Deep Dive into Java Project Loom - Swan Software Solutions. Swan Software Solutions. URL: https://swansoftwaresolutions.com/a-deep-dive-into-java-project-loom/

2. Zuko. Spring Boot Virtual Threads (java 21): setup, performance, and pitfalls. Spring Boot 123. URL: https://www.springboot-123.com/en/blog/spring-boot-virtual-threads-java21-guide/

3. Veen R., Vlijmincx D. Virtual Threads in Java. Berkeley, CA : Apress, 2024. URL: https://doi.org/10.1007/979-8-8688-0181-5.