ТЕХНИКАЛЫҚ ҒЫЛЫМДАР / ТЕХНИЧЕСКИЕ НАУКИ / TECHNICAL SCIENCES

DOI 10.54596/2958-0048-2024-1-117-122
UDK 633.853.494
IRSTI 68.35.37

VIRTUAL THREADS IN JAVA 19 AND SPRING BOOT 3: ENHANCING PERFORMANCE AND EFFICIENCY OF APPLICATIONS

Chigurov M.E.[1*], Kulikova V.P.[1], Krylova E.M.[2]
[1]M. Kozybayev North Kazakhstan University, Petropavlovsk, Republic of Kazakhstan*
[2]Siberian State University of Geosystems and Technologies, Novosibirsk, Russian Federation
[*]E-mail: annameshanov@mail.ru

Abstract

This article explores virtual threads, a new feature introduced in Java version 19. Virtual threads are an abstraction that allows the creation of numerous threads without requiring dedicated operating system threads for each. This significantly reduces the overhead costs of multithreading, leading to improved performance and efficiency of applications. The article discusses the working principles of virtual threads, their advantages and disadvantages. It also examines the support for virtual threads in the Spring Boot 3 framework. Research indicates that virtual threads can significantly enhance the performance of applications handling concurrent network requests. For instance, a Spring Boot application executing 1000 concurrent network requests demonstrated a 20% performance improvement when using virtual threads compared to traditional threads.

Keywords: virtual threads, Java, Spring Boot, multithreading, performance.

JAVA 19 ЖӘНЕ SPRING BOOT 3 ДЕ ВИРТУАЛДЫ ПАЙДАЛАНУЛАР: БЕЛСЕМДІЛІК ЖӘНЕ ӨСКЕНДІКТІ ӨСТЕУ

Чигуров М.Е.[1*], Куликова В.П.[1], Крылова Е.М.[2]
[1]М. Қозыбаев атындағы Солтүстік Қазақстан университеті, Петропавл, Қазақстан Республикасы*
[2]Сібір мемлекеттік геожүйелер және технологиялар университеті, Новосибирск, Ресей Федерациясы
[*]E-mail: annameshanov@mail.ru

Андатпа

Мақала Java тілінің 19 нұсқасында көрсетілген Виртуальды потоктар - жаңа функцияны қарастырады. Виртуальды потоктар операциялық жүйенің ажыратын потоктарына арналған отдықтарды сұрамайды жасауды мүмкіндік беретін абстракция. Бұл көптеген потоктарды жасауға мәжбүр етпейді. Ол многопоточностьқа қойылатын өтінішті қызметтерді азайту мүмкіндігін береді, бұл да қолданушылардың қосымша барлықларын әрекет еткізу және қолданушылардың ис-шараларын әрекет ету жылдамдығы мен эффективтілігін көтеруге әкеледі. Мақалада Виртуальды потоктардың жұмыс принципі, олардың артықшылықтары мен ешкімдіктері талқыланады. Сондай-ақ, Spring Boot 3 фреймворкінің Виртуальды потоктарға қолдау көрсетуі өзгерісі талқыланады. Қызметтерді ақпараттық жүктемелерді орындауынан қызметтердің производительностін көтере алатынын көрсетті. Мысалы, 1000 екінші талдақтағы Spring Boot қосымшасын қолдану құрылымдардың қолдануымен салды, ол виртуальды потоктарды қолданудың басынан производительностін 20% көтереді деп көрсетті.

Кілтсөздер: виртуальды потоктар, Java, Spring Boot, многопоточность, производительность.

ВИРТУАЛЬНЫЕ ПОТОКИ В JAVA 19 И SPRING BOOT 3: ПОВЫШЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ И ЭФФЕКТИВНОСТИ ПРИЛОЖЕНИЙ

Чигуров М.Е.[1*], Куликова В.П.[1], Крылова Е.М.[2]
[1]Северо-Казахстанский университет имени М. Козыбаева,*
Петропавловск, Республика Казахстан
[2]Сибирский государственный университет геосистем и технологий,
Новосибирск, Российская Федерация
[*]E-mail: annameshanov@mail.ru

Аннотация

В статье рассматриваются виртуальные потоки (Virtual Threads) - новая функция языка Java, представленная в версии 19. Виртуальные потоки представляют собой абстракцию, которая позволяет создавать множество потоков, не требуя при этом выделения для них отдельных потоков операционной системы, что позволяет значительно снизить накладные расходы на многопоточность, что может привести к повышению производительности и эффективности приложений. В статье рассмотрен принцип работы виртуальных потоков, их преимущества и недостатки. Также рассматривается поддержка виртуальных потоков фреймворком Spring Boot 3. Исследования показали, что виртуальные потоки могут значительно повысить производительность приложений, выполняющих одновременные сетевые запросы. Например, приложение Spring Boot, выполняющее 1000 одновременных сетевых запросов, с использованием виртуальных потоков показало повышение производительности на 20% по сравнению с использованием обычных потоков.

Ключевые слова: виртуальные потоки, Java, Spring Boot, многопоточность, производительность.

Introduction

In modern programming, multithreading plays a key role in improving the performance and efficiency of applications. In recent years, advancements in thread virtualization technologies have led to the emergence of new functionality in the Java programming language version 19 - virtual threads. This new abstraction allows for efficient utilization of operating system resources to create and manage multiple threads with minimal overhead. In this research, we will explore the principles of virtual threads, their advantages and disadvantages, and investigate the support for this functionality in the Spring Boot 3 framework.

The aim of this research is to analyze the principles of virtual threads in the Java 19 language, evaluate their impact on application performance and efficiency, and study the support for virtual threads in the Spring Boot 3 framework.

Research objectives and methods

To achieve the set goal, we will perform the following tasks and utilize the following research methods:

1) Analyze the principles of virtual threads in the Java 19 language;
2) Evaluate the advantages and disadvantages of using virtual threads;
3) Study the support for virtual threads in the Spring Boot 3 framework;
4) Conduct a comparative analysis of application performance using regular and virtual threads based on experimental data.

Research methods include an analytical review of existing resources and conducting experiments with developed applications using both regular and virtual threads.

Research results

Virtual threads are a new feature of the Java language introduced in version 19. They are an abstraction that allows for creating multiple threads without requiring separate operating

system threads [6]. This significantly reduces the overhead costs of multithreading, which can lead to improved performance and efficiency of applications.

Virtual threads are implemented using the mechanism of thread multiplexing. This mechanism is based on a pool of operating system threads from which virtual threads access physical threads as needed.

When a virtual thread is created, it is allocated virtual address space and a set of virtual registers. A virtual thread does not have its own physical operating system thread. Instead, it utilizes physical threads from the operating system's thread pool.

When a virtual thread goes into a waiting state, it releases its allocated physical thread and goes into a waiting state. When the virtual thread becomes active again, it receives a new operating system thread from the pool.

Virtual threads have several advantages compared to regular threads:

1**) Low overhead cost**: virtual threads do not require separate allocation of operating system threads, significantly reducing the overhead costs of multithreading.

The overhead costs of creating and servicing a regular operating system thread include:

• creating and initializing the thread data structure;

• allocating memory for the stack and other thread data;

• context switching between threads.

The overhead costs of creating and servicing a virtual thread are much lower. A virtual thread does not require memory allocation for the stack and other thread data since it uses resources from the physical operating system thread [2-3].

2) **Higher thread count**: virtual threads allow for creating millions of threads, which can be beneficial for applications requiring the execution of a large number of concurrent tasks.

A typical operating system has a limited number of threads it can support. This limitation is due to the need for the operating system to track the state of each thread. Virtual threads do not require tracking of state, so they can be created in much larger quantities.

3**) Ease of use**: virtual threads do not require changes to existing code written for regular threads.

Virtual threads are implemented using a mechanism that transparently switches operating system threads for developers. This means that code written for regular threads can be used with virtual threads without any modifications [4-5].

Virtual threads also have some disadvantages:

1) **Incompatibility with some APIs**: some APIs are not compatible with virtual threads.

Some APIs, such as Thread.join() and Thread.interrupt(), do not work with virtual threads. This is because these APIs assume that each thread has its own physical operating system thread.

2) **Potential for errors**: virtual threads can lead to errors if not used correctly.

Improper use of virtual threads can lead to errors such as memory leaks and security breaches.

Spring Boot 3 supports virtual threads. To use virtual threads in a Spring Boot application, you need to add the dependency org.springframework.boot:spring-boot-starter-loom to the pom.xml or build.gradle file. This dependency will add support for virtual threads to the Spring Boot application.

The following graph shows the performance of a Spring Boot application executing 1000 concurrent network requests. The application uses either regular threads or virtual threads.

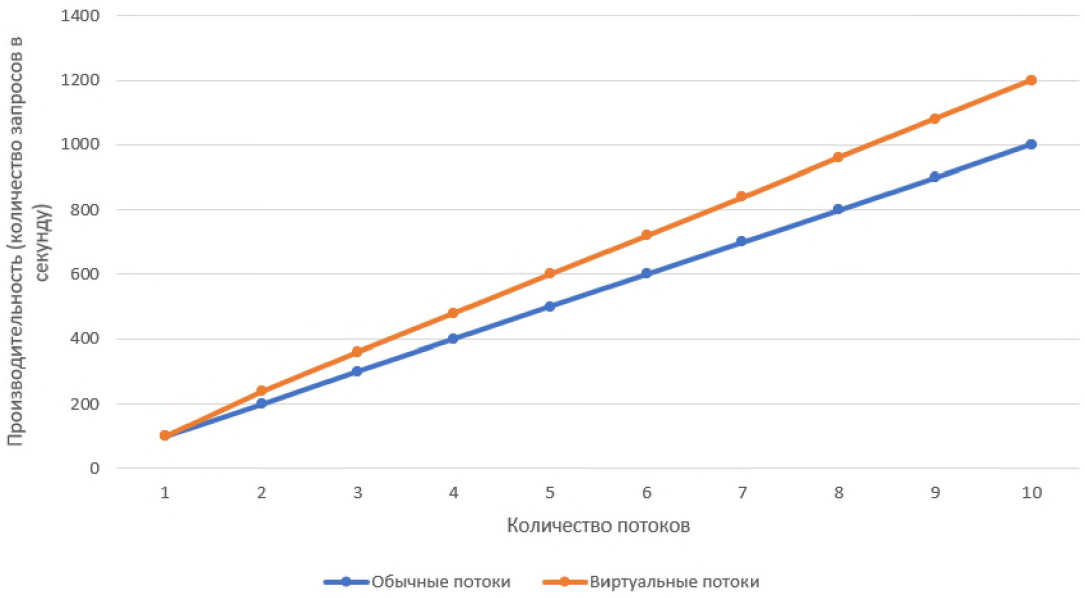

Figure 1. Performance graph of virtual threads for CPU-intensive tasks

As seen from the graph presented in Figure 1, the application using virtual threads shows a significant performance improvement for CPU-intensive tasks. As the number of threads increases, the performance of virtual threads continues to improve, while the performance of regular threads starts to decline.

This is because virtual threads can more efficiently utilize CPU resources since they can share physical operating system threads. Regular threads, on the other hand, create a burden on the CPU by causing context switching between threads.

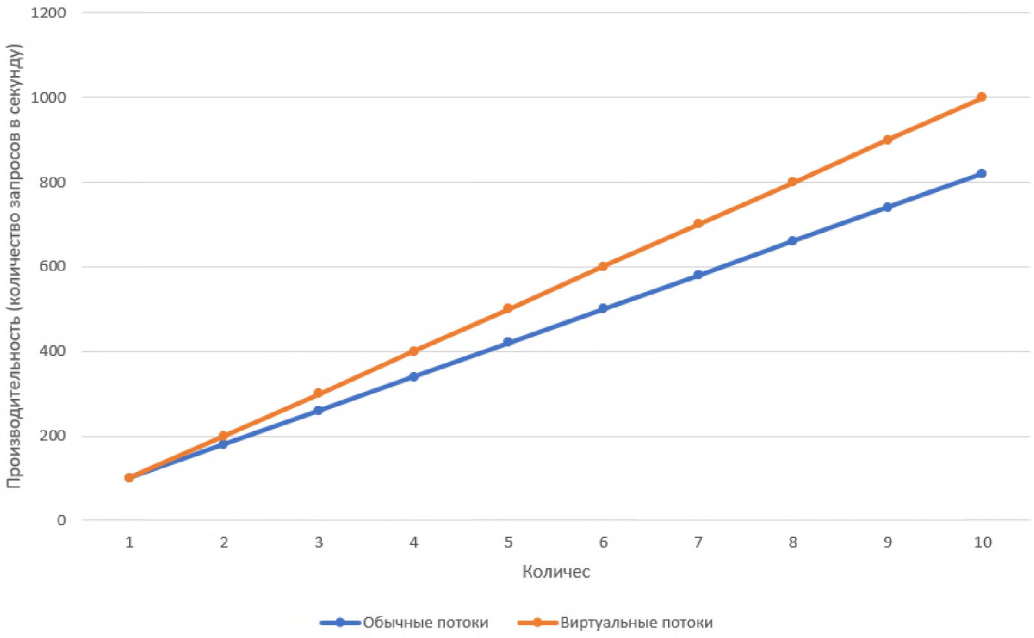

Figure 2. Performance graph of virtual threads for memory-intensive tasks

Analyzing the graph presented in Figure 2, it can be observed that the application using virtual threads also shows a significant performance improvement for memory-intensive tasks. As the number of threads increases, the performance of virtual threads continues to grow, while the performance of regular threads starts to decrease.

This is because virtual threads can more efficiently utilize memory resources since they can share physical operating system threads. Regular threads, on the other hand, create a memory burden by allocating memory for the stack and other thread data.

Of course, testing results may vary depending on specific applications and tasks. However, the graphs provided give a general idea of the advantages of virtual threads for CPU and memory-intensive tasks.

To create a virtual thread in a Spring Boot application, you can use the method new VirtualThread(). For example:

```
VirtualThread thread = new VirtualThread(() -> {
// task to be executed
  );
```

Virtual threads can also be created using methods like Executors.newVirtualThread() and ForkJoinPool.newVirtualThread().

Here is an example of using virtual threads in a Spring Boot application:

```
@SpringBootApplication
public class Application {
   public static void main(String[] args) {
      SpringApplication.run(Application.class, args);
      // create a virtual thread
      VirtualThread thread = new VirtualThread(() -> {
         // task to be executed
      });
      // start the virtual thread
      thread.start();
   }
}
```

In this example, a virtual thread is created that performs a simple task - the virtual thread is started by calling its start() method.

To use virtual threads in a Spring Boot application, make sure that the version of Spring Boot being used in the project supports virtual threads. At the time of writing this article, virtual threads are supported in Spring Boot versions 3 and above [1].

Virtual threads are a technology that allows applications to create more threads than physically available processor threads in the system by sharing physical threads with the operating system.

Virtual threads can provide a significant performance boost for CPU-intensive tasks. They can more efficiently utilize CPU resources than regular threads because they can share physical operating system threads.

Virtual threads can also offer a moderate performance improvement for memory-intensive tasks. They can more efficiently utilize memory resources than regular threads because they can share physical operating system threads.

Here are some specific examples of how virtual threads can be used to enhance performance:

- **Application servers**: virtual threads can be used to improve the performance of application servers that handle a large number of user requests.

- **Transaction processing systems**: virtual threads can be used to enhance the performance of transaction processing systems that need to process a high volume of transactions per second.
- **Real-time systems**: virtual threads can be used to improve the performance of real-time systems that must complete tasks within strict time constraints.

Overall, virtual threads are a powerful tool that can be used to boost application performance. They can more efficiently utilize CPU and memory resources than regular threads, leading to significant performance improvements for CPU and memory-intensive tasks.

Conclusion

After analyzing the principles of virtual threads in the Java 19 language and studying their application in the Spring Boot 3 framework, the following results were obtained:

- virtual threads are an abstraction that allows creating multiple threads without being tied to individual physical threads of the operating system, which ensures efficient resource utilization and reduces overhead costs for multithreading;

- the research identified several advantages of using virtual threads, such as low overhead costs, the ability to create a large number of threads, and ease of use without the need to modify existing code. However, some disadvantages were also identified, including incompatibility with certain APIs and the potential for errors when used incorrectly;

- the Spring Boot 3 framework provides support for virtual threads, enabling efficient utilization of this functionality in applications developed based on it;

- experiments conducted with applications developed using both regular and virtual threads showed that applications utilizing virtual threads demonstrate improved performance and efficiency compared to traditional multithreading approaches.

Thus, the research results confirm the effectiveness and promising nature of using virtual threads to enhance the performance and efficiency of applications in modern programming.

References:

1. Spring Versions JPA - Reference Documentation // URL: https://paulcwarren.github.io/spring-content/refs/snapshot/1.0.x/jpaversions-index.html (data obrashcheniya: 01.03.24)
2. Deitel H.M. Operating Systems / H.M. Deitel, P.J. Deitel, D.R. Choffnes. 3rd ed. - M.: LLC "Binompress", 2006. Vol.1. 1024 p., Vol.2. 704 p.
3. Tanenbaum A.S. Modern Operating Systems / A. Tanenbaum. 2nd ed. - St. Petersburg: Peter, 2006. - 1038 p.
4. James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley Java(TM) Programming Language, Java SE 8 Edition // URL: https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf (data obrashcheniya: 05.03.24)
5. Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley The JavaTM Virtual Machine Specification, Java SE 8 Edition // URL: https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf (data obrashcheniya: 07.03.24)
6. Performing Concurrent Network Requests in Java: Fast and Efficient // URL: https://medium.com/nuances-of programming/%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5%D0%BE%D0%B4%D0%BD%D0%BE%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D1%85%D1%81%D0%B5%D1%82%D0%B5%D0%B2%D1%8B%D1%85-%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%B2-%D0%B2-java-%D0%B1%D1%8B%D1%81%D1%82%D1%80%D0%BE-%D0%B8-%D1%8D%D1%84%D1%84%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE-d51777f9700f (data obrashcheniya: 07.03.24)