PERFORMANCE COMPARISON OF JAVA AND C++ IN HIGH-LOAD SERVER APPLICATIONS DEVELOPMENT

Smirnov A.
master's degree, Perm National Research Polytechnic University,
Perm, Russia

СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ JAVA И C++ В РАЗРАБОТКЕ ВЫСОКОНАГРУЖЕННЫХ СЕРВЕРНЫХ ПРИЛОЖЕНИЙ

Смирнов А.В.
магистр, Пермский национальный исследовательский
политехнический университет,
Пермь, Россия
https://doi.org/10.5281/zenodo.15220801

Abstract

This article presents a comparative analysis of the performance of the Java and C++ programming languages in the development of high-load server applications. Special attention is given to key aspects affecting computational efficiency, such as memory management, multithreading, and code optimization. The impact of JIT compilation on code adaptation to real execution conditions is examined. Examples of optimizations, including inlining, escape analysis, and branch prediction, are considered, along with their role in improving program execution speed. The advantages and limitations of both languages in the context of server computing are identified.

Аннотация

В данной статье проводится сравнительный анализ производительности языков программирования Java и C++ в разработке высоконагруженных серверных приложений. Особое внимание уделяется ключевым аспектам, влияющим на вычислительную эффективность, таким как управление памятью, многопоточность и оптимизация кода. Исследуется влияние JIT-компиляции на адаптацию кода к реальным условиям выполнения. Рассматриваются примеры оптимизаций, таких как inlining, escape analysis и прогнозирование ветвлений, а также их роль в повышении скорости выполнения программ. Выявляются преимущества и ограничения обоих языков в контексте серверных вычислений.

Keywords: Java, C++, performance, high-load server applications, memory management, multithreading, JIT compilation, static compilation.

Ключевые слова: Java, C++, производительность, высоконагруженные серверные приложения, управление памятью, многопоточность, JIT-компиляция, статическая компиляция.

Введение

На фоне стремительного роста объемов обрабатываемых данных и увеличения требований к отказоустойчивости серверных систем выбор языка программирования становится одним из ключевых факторов, определяющих производительность и масштабируемость приложений. Java и C++ это два широко используемых языка, применяемых для разработки высоконагруженных серверных решений, каждый из которых предлагает уникальный подход к управлению ресурсами. Их различия в организации работы с памятью, многопоточностью и механизмами компиляции оказывают значительное влияние на вычислительную эффективность, что делает их сравнение особенно актуальным.

Современные серверные приложения должны грамотно использовать вычислительные мощности и адаптироваться к изменяющимся условиям нагрузки, при этом обеспечивая стабильную и предсказуемую работу. В этом контексте важно учитывать, какие механизмы управления памятью и потоками предлагает язык, а также насколько хорошо он способен оптимизировать выполнение кода. Разные принципы компиляции и оптимизационные стратегии Java и C++ влияют на скорость выполнения программ, потребление ресурсов и возможности масштабирования, что требует детального анализа их преимуществ и ограничений в реальных условиях эксплуатации.

Цель данной статьи – оценить и сопоставить производительность Java и C++ в контексте высоконагруженных серверных приложений, выявить сильные и слабые стороны каждого подхода. Актуальность исследования обусловлена растущей конкуренцией между технологиями и необходимостью выбора оптимальных инструментов для реализации особенно важных приложений. Методы исследования включают сравнительный анализ архитектурных особенностей, экспериментальное тестирование и анализ существующих бенчмарков, что позволяет оценить производительность рассматриваемых языков.

Основная часть. Сравнительный анализ управления памятью и многопоточности в Java и C++

Эффективность работы высоконагруженных серверных приложений во многом определяется тем, как язык программирования обрабатывает память и организует многопоточное выполнение кода. Java и C++ предлагают разные подходы к

этим ключевым аспектам, что оказывает значительное влияние на производительность.

Одно из наиболее значимых различий между Java и C++ заключается в **способе управления памятью**. В Java используется автоматический сборщик мусора (Garbage Collector, GC), который освобождает память, занятую объектами, на которые нет ссылок [1]. Это снижает вероятность утечек памяти и упрощает разработку, так как IT-специалисту не нужно явно освобождать ресурсы. Однако автоматическая сборка мусора приводит к дополнительным накладным расходам и может приводить к паузам в выполнении программы, особенно в условиях высокой нагрузки. Современные алгоритмы GC, такие как G1 и ZGC, минимизируют задержки, но не устраняют их полностью.

В C++ управление памятью возложено на программиста, что предоставляет больший контроль над распределением и освобождением ресурсов. Этот подход позволяет избежать накладных расходов, связанных с работой GC, и гарантировать более предсказуемое использование оперативной памяти. Однако ошибки управления памятью, такие как утечки, разыменование нулевого указателя и использование освобожденной памяти, могут приводить к серьезным сбоям в работе приложения. Для упрощения управления памятью в C++ широко используются механизмы **RAII** (Resource Acquisition Is Initialization) и **умные указатели** (std::unique_ptr, std::shared_ptr), которые частично автоматизируют процесс работы с ресурсами.

Разработка многопоточных программ требует эффективных механизмов управления потоками и синхронизации. Java предоставляет высокоуровневую модель работы с потоками через **java.util.concurrent**, что позволяет использовать **пулы потоков (ThreadPoolExecutor)**, **асинхронные задачи (CompletableFuture)** и другие инструменты для эффективной работы с параллельными вычислениями. Основное преимущество Java заключается в переносимости многопоточных решений: виртуальная машина (JVM) абстрагирует реализацию потоков от конкретной операционной системы, обеспечивая кроссплатформенную стабильность. Хотя Java предлагает удобные инструменты для работы с потоками, бенчмарки показывают, что в задачах с интенсивными операциями чтения и записи ее производительность может уступать C++ из-за накладных расходов на управление потоками и JVM (рис. 1).

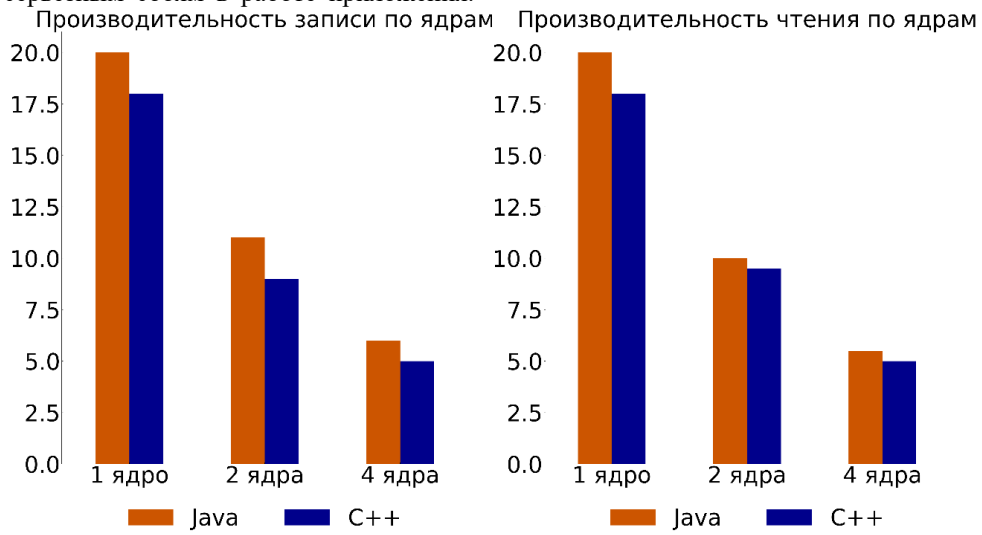

Рисунок 1. Сравнение производительности Java и C++ в многопоточных операциях [2]

В C++ стандартная библиотека предлагает более низкоуровневый механизм работы с потоками через **std::thread**, **std::mutex** и **std::atomic**. Хотя это обеспечивает большую гибкость, разработчик вынужден самостоятельно управлять созданием и завершением потоков. Для упрощения работы с многопоточностью применяются библиотеки, такие как **Intel Threading Building Blocks (TBB)** и **OpenMP**, которые реализуют высокоуровневые механизмы управления параллельными вычислениями.

Разница между Java и C++ также проявляется в модели памяти и способах синхронизации. В Java применяется модель **Java Memory Model (JMM)**, которая строго определяет поведение потоков при доступе к разделяемым данным и гарантирует предсказуемость выполнения [3]. В C++ аналогичная модель появилась только в стандарте C++11, но она предоставляет более низкоуровневый контроль за работой потоков, что может дать прирост производительности в критически важных сценариях.

Существенным преимуществом Java является **автоматическое управление потоками на уровне виртуальной машины**, что позволяет JVM оптимизировать работу многопоточных приложений в зависимости от нагрузки. В C++ многопоточность более эффективна в случаях, когда важно минимизировать накладные расходы, так как отсутствуют промежуточные абстракции, свойственные JVM.

В многопоточных средах Java предлагает удобные инструменты для конкурентного программирования, что снижает сложность разработки. Однако накладные расходы, связанные с управлением потоками и синхронизацией, могут сделать Java менее эффективной в высокопроизводительных вы-

числительных задачах. C++ требует большего объема кода для реализации многопоточных решений, но предоставляет больше возможностей для низкоуровневой оптимизации, что особенно важно в распределенных системах, таких как микросервисы [4].

Выбор между Java и C++ зависит от конкретного сценария использования. Если критична предсказуемость работы с памятью и низкие задержки, C++ может быть предпочтительнее. Если важны простота разработки, кроссплатформенность и удобная работа с многопоточностью, Java становится более подходящим выбором.

Оптимизация кода и влияние JIT-компиляции на производительность

Производительность серверных приложений во многом зависит от эффективности выполнения машинного кода. Java и C++ используют разные подходы к компиляции и оптимизации, что напрямую влияет на скорость выполнения программ, адаптацию к нагрузкам и общее потребление ресурсов.

Так C++ использует **статическую компиляцию**, при которой весь код компилируется в исполняемый файл до запуска программы. Это позволяет выполнять различные оптимизации на этапе компиляции, такие как разворачивание циклов, удаление неиспользуемого кода и агрессивное inline-встраивание функций. Код C++ напрямую компилируется в машинные инструкции процессора, что даёт возможность использовать специфичные для архитектуры CPU оптимизации, такие как векторизация и предсказание ветвлений.

В отличие от этого, Java применяет **динамическую (Just-In-Time, JIT) компиляцию**, что означает, что код сначала транслируется в байт-код, а затем JVM во время выполнения преобразует его в машинные инструкции [5]. Такой подход позволяет адаптировать исполняемый код под реальные условия работы, применять профильные оптимизации и избегать статических ограничений. JIT-компиляторы, такие как HotSpot, способны анализировать участки кода и переоптимизировать их в процессе работы приложения.

Основное различие между этими подходами заключается в том, что в C++ программа оптимизируется один раз перед выполнением, а в Java – динамически, на основе реальных данных о ее работе. Это дает JIT-компилятору возможность применять агрессивные оптимизации, которые невозможно реализовать статически. Однако в Java есть дополнительный этап интерпретации байт-кода перед компиляцией, что приводит к увеличению времени старта программы.

JIT-компилятор в Java применяет ряд техник, которые позволяют ему в ряде случаев достичь или даже превзойти по производительности статически скомпилированный код C++. Одними из ключевых технологий являются адаптивная компиляция, динамическое inlining-встраивание функций и другие (таблица 1).

Таблица 1.

Влияние JIT-оптимизаций на производительность Java [6, 7]

Оптимизация JIT	Описание	Влияние на производительность
Inlining	Замена вызова функции ее телом.	Уменьшает накладные расходы на вызовы функций.
Escape analysis	Определение объектов, не покидающих метод, для размещения в стеке.	Снижает нагрузку на GC, ускоряет выполнение.
Branch prediction	Перестановка кода на основе реальных данных исполнения.	Повышает эффективность выполнения условных операторов.
Adaptive compilation	Постепенное улучшение кода во время работы программы.	Долговременный прирост производительности.
Deoptimization	Возвращение к менее оптимизированному коду при изменении условий.	Повышает стабильность работы программы, снижая вероятность деградации производительности

В C++ аналогичная оптимизация escape analysis возможна только на этапе статической компиляции, однако для ее реализации требуется явное использование механизма **placement new** или ручное управление памятью. Что касается deoptimization (обратной деоптимизации), в C++ такой механизм отсутствует, так как код оптимизируется заранее и не может изменяться во время выполнения. Хотя C++ традиционно считается более производительным языком благодаря отсутствию накладных расходов на виртуальную машину и динамическую компиляцию, в некоторых случаях Java также может демонстрировать высокую производительность и обеспечивать конкурентоспособные результаты.

Во-первых, в приложениях с долгим временем выполнения и повторяющимися вычислениями JIT-компилятор способен со временем значительно ускорить выполнение кода. В таких системах JIT анализирует реальные данные и подстраивает оптимизации в зависимости от нагрузки, в то время как в C++ код остаётся неизменным после компиляции.

Во-вторых, работа с многопоточностью в Java может быть более эффективной за счет того, что JVM оптимизирует переключение потоков и управляет их состоянием на уровне байт-кода. В C++ программист вынужден самостоятельно учитывать особенности платформы, на которой запускается приложение, что усложняет создание универсального многопоточного кода.

Третьим важным аспектом является оптимизация кэш-памяти и прогнозирование ветвлений. JIT-компилятор может перестраивать код таким образом, чтобы он лучше соответствовал конкретной

архитектуре процессора. В C++ такие оптимизации выполняются компилятором заранее и не могут изменяться во время работы программы.

Наконец, стоит отметить, что современные интерпретаторы JVM поддерживают предварительную компиляцию (AOT, Ahead-of-Time), что позволяет сократить время старта приложения [8]. Это снижает традиционный недостаток Java – медленный запуск, делая ее конкурентоспособной с C++ в задачах, требующих быстрой инициализации.

Java и C++ представляют два разных подхода к компиляции и оптимизации кода. Статическая компиляция C++ позволяет заранее достичь высокой производительности за счёт глубоких оптимизаций на этапе компиляции. Однако JIT-компиляция в Java дает возможность динамически подстраиваться под реальные условия работы, что позволяет оптимизировать выполнение кода в зависимости от текущей нагрузки.

Выводы

Сравнение производительности Java и C++ в разработке высоконагруженных серверных приложений показывает, что каждый язык обладает своими преимуществами и ограничениями. C++ обеспечивает высокую производительность за счет статической компиляции, низкоуровневого управления памятью и возможности тонкой настройки программного кода. Это делает его предпочтительным выбором для систем, где критически важна минимизация накладных расходов и предсказуемость работы. В то же время, Java предлагает мощные механизмы динамической оптимизации, включая JIT-компиляцию и автоматическое управление памятью.

Выбор между этими языками зависит от требований к производительности, гибкости и удобству поддержки кода. C++ лучше подходит для задач, требующих максимального контроля над ресурсами, таких как разработка системных программ, игр и высокопроизводительных серверных компонентов. Java же показывает высокую эффективность в длительно работающих приложениях с изменяющейся нагрузкой, например в корпоративных и облачных системах. При проектировании высоконагруженных серверных решений важно учитывать как базовые показатели скорости выполнения, так и архитектурные особенности языка, влияющие на долговременную масштабируемость и устойчивость системы.

Список литературы:

1. Назарян А.К., Карцан И.Н. Сравнительный анализ языков программирования C++ и Java с точки зрения обеспечения безопасности кода // Современные инновации, системы и технологии. 2024. Т. 4(4). С. 0186-98.

2. Brandefelt L., Heyman H. A Comparison of Performance & Implementation Complexity of Multithreaded Applications in Rust, Java and C++ // School of Electrical Engineering and Computer Science, KTH Institute of Technology, Stockholm, Sweden. 2020. 19 p.

3. Moiseenko E., Podkopaev A., Koznov D. A survey of programming language memory models // Programming and Computer Software. 2021. Vol. 47. P. 439-56. DOI: 10.1134/S0361768821060050 EDN: PQZNIE

4. Bolgov S. Optimizing microservices architecture performance in fintech projects // Bulletin of the Voronezh Institute of High Technologies. 2025. Vol. 19(1). URL: https://vestnikvivt.ru/ru/journal/pdf?id=1401

5. Zhao Y., Damevski K., Chen H. A systematic survey of just-in-time software defect prediction // ACM Computing Surveys. 2023. Vol. 55(10). P. 1-35. DOI: 10.1145/3561818 EDN: JLIDBD

6. Постнов С.С. Обзор технологий JIT-компиляции // International Journal of Open Information Technologies. 2020. Т. 8(9). С. 8-17. EDN: UFNVMC

7. Bernhard L., Scharnowski T., Schloegel M., Blazytko T., Holz T. JIT-picking: Differential fuzzing of JavaScript engines // Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security 2022. P. 351-364. DOI: 10.1145/3548606.3560624

8. Blazhkovskii A. Optimization of mobile application performance: modern approaches and methods // ISJ Theoretical & Applied Science. 2024. Vol. 140(12). P. 290-294. DOI: 10.15863/TAS.2024.12.140.35