Программные системы и вычислительные методы
Правильная ссылка на статью:
Макаров И.С., Ларин Д.В., Воробьева Е.Г., Емелин Д.П., Карташов Д.А. Влияние асинхронных и многопоточных моделей обработки запросов на производительность серверных веб-приложений // Программные системы и вычислительные методы. 2025. № 1. DOI: 10.7256/2454-0714.2025.1.73665 EDN: UZPPWT URL: https://nbpublish.com/library_read_article.php?id=73665

Влияние асинхронных и многопоточных моделей обработки запросов на производительность серверных веб-приложений

Макаров Игорь Сергеевич

ORCID: 0009-0004-8734-2667

кандидат технических наук

зав. кафедрой; кафедра прикладной информатики (ПИ); Поволжский государственный университет телекоммуникаций и информатики

443010, Россия, Самарская обл., г. Самара, Самарский р-н, ул. Льва Толстого, д. 23

igor-psati@yandex.ru

Ларин Денис Вячеславович

ORCID: 0009-0000-1904-7210

студент; кафедра информатики и вычислительной техники (ИВТ); Поволжский государственный университет телекоммуникаций и информатики

443010, Россия, Самарская область, г. Самара, ул. Льва Толстого, 23

denlar1989@gmail.com

Воробьева Евгения Григорьевна

ORCID: 0009-0008-8225-7091

студент; кафедра информатики и вычислительной техники (ИВТ); Поволжский государственный университет телекоммуникаций и информатики

443010, Россия, Самарская обл., г. Самара, Самарский р-н, ул. Льва Толстого, д. 23

vorobeva.g2004@gmail.com

Емелин Даниил Павлович

студент; кафедра информатики и вычислительной техники (ИВТ); Поволжский государственный университет телекоммуникаций и информатики

443010, Россия, Самарская область, г. Самара, ул. Льва Толстого, 23

demelin163@gmail.com

Карташов Дмитрий Александрович

студент; кафедра информатики и вычислительной техники (ИВТ); Поволжский государственный университет телекоммуникаций и информатики

443010, Россия, Самарская обл., г. Самара, Самарский р-н, ул. Льва Толстого, д. 23

tawerka40@gmail.com

Статья из рубрики "Языки программирования"

DOI:

10.7256/2454-0714.2025.1.73665

EDN:

UZPPWT

Дата направления статьи в редакцию:

12-03-2025

Аннотация: Объектом исследования являются серверные веб-приложения и их производительность при обработке большого количества одновременных запросов. В качестве предмета исследования рассматриваются асинхронные технологии (Node.js, Python Asyncio, Go, Kotlin Coroutines) и многопоточные модели (Java Threading, Python Threading). Авторы подробно анализируют асинхронные циклы событий, горутины, корутины и классические многопоточные подходы, оценивая их эффективность в задачах с интенсивным использованием I/O и вычислительных ресурсов. Проводится эксперимент с разработкой API на трёх языках (Java, Node.js, Go) и тестированием при помощи утилиты hey. Также исследуются особенности масштабируемости, оптимизации производительности, использование кэширования, обработка ошибок, нагрузочные тесты и особенности реализации параллельных вычислений. Цель исследования — определить, какие подходы обеспечивают наибольшую производительность в серверных приложениях. Методы исследования включают нагрузочное тестирование, сбор метрик (время отклика, пропускная способность и потребление ресурсов сервера) и анализ результатов. Научная новизна заключается в сравнении асинхронных и многопоточных методов в реальных сценариях веб-разработки. Основными выводами исследования являются рекомендации по использованию асинхронных технологий в высоконагруженных I/O задачах и многопоточности в вычислительно сложных сценариях. Полученные результаты помогут разработчикам оптимизировать производительность серверных приложений в зависимости от их задач и нагрузки. Дополнительно исследование рассматривает аспекты сложности отладки асинхронных приложений, влияние пулов потоков на производительность многопоточных решений, а также сценарии, в которых асинхронные и многопоточные подходы могут дополнять друг друга. Особое внимание уделено управлению ресурсами сервера при масштабируемых нагрузках, что позволит IT-специалистам более точно выбирать инструменты и технологии для решения конкретных задач. В заключении обсуждаются возможные пути оптимизации работы серверных приложений, включая использование новых подходов и алгоритмов, а также перспективы развития асинхронных и многопоточных технологий в контексте высоконагруженных систем, их влияние на общую архитектуру приложений, а также на повышение отказоустойчивости и безопасности.

Ключевые слова:

асинхронность, многопоточность, производительность, серверные веб-приложения, нагрузочное тестирование, Node.js, Python Asyncio, Go горутины, Java Threading, производительность серверных приложений

Введение

Современные серверные веб-приложения сталкиваются с растущими требованиями к обработке большого количества одновременных запросов. С увеличением числа пользователей и сложностью операций, особенно в приложениях, работающих с API, базами данных и внешними сервисами, производительность серверной части становится критически важной. В этом контексте выбор между асинхронными и многопоточными подходами приобретает большое значение.

Асинхронные технологии, такие как Node.js (цикл событий), Python Asyncio (параллелизм, управляемые события), Go (горутины), Kotlin Coroutines, предлагают возможности для улучшения производительности за счёт неблокирующих операций ввода-вывода и лёгкости управления большим числом одновременных соединений. С другой стороны, многопоточные модели, такие как Java Threading и Python Threading, обеспечивают надёжность и простоту логики, но могут иметь накладные расходы при создании большого числа потоков. [1, с.93-95],[2]

В данной статье рассматривается исследование производительности серверных веб-приложений с использованием асинхронных и многопоточных подходов.

2. Обзор технологий и подходов

Асинхронный цикл событий.

Асинхронный цикл событий представлен такими технологиями, как Node.js и Python Asyncio. Данный подход предполагает использование одного потока для управления большим количеством запросов через неблокирующий ввод-вывод. Это позволяет достичь высокой производительности при операциях ввода-вывода, однако усложняет отладку и требует использования асинхронных библиотек. [5]

Горутины и корутины.

Go и Kotlin Coroutines используют легковесные потоки, управляемые рантаймом языка. Они обеспечивают высокую параллельность и особенно эффективны в ресурсоемких задачах. Однако разработчику необходимо явно указывать конкурентные вычисления. [3, с.260-261],[4]

Многопоточное программирование.

Многопоточная модель (Java, Python Threading) выделяет отдельный поток для каждого запроса или использует пул потоков. Этот подход полезен для задач, требующих параллельной обработки, однако она требует внимательного подхода к проектированию и синхронизации, чтобы избежать ошибок и потерь производительности. [1, с.93-95]

Ниже приведена сравнительная таблица 2.1 данных технологий:

Таблица 2.1 – сравнение технологий и подходов

Подход	Технологии	Принцип работы	Преимущества	Недостатки

Асинхронный цикл событий	Node.js, Python Asyncio	Один поток управляет большим количеством запросов через неблокирующий ввод-вывод	Высокая производительность при I/O операциях	Сложность отладки, необходимость адаптации асинхронным библиотекам
Горутины и корутины	Go, Kotlin Coroutines	Легковесные потоки, управляемые рантаймом языка	Высокая параллельность, эффективность в ресурсоемких задачах	Требует понимания конкурентного программирован
Многопоточное программирование	Java Threading, Python Threading	Каждый запрос получает отдельный поток (или пул потоков)	Хорошая поддержка в языках, простота логики	Большие накладные расходы создание потоко блокировки ресурсов

3. Методология исследования

Для проведения исследования было разработано серверное приложение с API на трёх языках: Java (многопоточное), Node.js (асинхронное), Go (горутины). API-эндпоинт выполняет обработку входящего запроса, чтение данных из базы (эмуляция задержки) и отправку ответа. Нагрузочные тесты проводились с разным количеством одновременных запросов (50, 100, 500, 1000, 5000). Оценивались среднее время ответа, пропускная способность и использование ресурсов сервера (CPU, RAM).

4. Реализация

Нагрузочные тесты проводились для разных уровней нагрузки: 50, 100, 500, 1000 и 5000 одновременных запросов, что позволило выявить, как каждый подход справляется с увеличением нагрузки. Для каждого сценария тестирования мы замеряли следующие параметры:

Среднее время ответа — среднее время, которое прошло с момента получения запроса сервером до отправки ответа клиенту. Важный показатель для оценки отзывчивости приложения (таблица 4.1).

Таблица 4.1 – среднее время ответа

Язык	**Низкая нагрузка (50 запросов)**	**Высокая нагрузка (5000 запросов)**
Java	~304-306 мс (задержка + накладные расходы)	~1094-8966 мс (рост из-за блокировки потоков)
Node.js	~236-248 мс (почти без накладных расходов)	~401-785 мс (event loop справляется хорошо)
Go	~212-216 мс (немного выше из-за планировщика)	~384-549 мс (эффективное управление горутинами)

Node.js и Go имеют более низкую латентность под высокой нагрузкой благодаря асинхронной обработке и лёгкости создания новых задач.

Пропускная способность (RPS) — количество запросов, которые сервер способен обработать за единицу времени (таблица 4.2).

Таблица 4.2 – пропускная способность

Язык	**Низкая нагрузка (50 запросов)**	**Высокая нагрузка (5000 запросов)**
Java	~304-306 мс (задержка + накладные расходы)	~380-9087мс (рост из-за блокировки потоков)
Node.js	~233-249 мс (почти без накладных расходов)	~355-800 мс (event loop справляется хорошо)
Go	~211-217 мс (немного выше из-за планировщика)	~343-560 мс (эффективное управление горутинами)

Go лидирует из-за лёгкости горутин и быстрого планировщика, за ним идёт Node.js. Java замедляется из-за затрат на управление потоками и переключение контекста.

Для проведения нагрузочного тестирования нами была использована утилита hey.

hey — это современный инструмент для нагрузочного тестирования HTTP-серверов, написанный на языке Go. Его можно считать современным аналогом таких утилит, как ApacheBench или wrk, но с рядом преимуществ:

- Лёгкость и эффективность: hey позволяет быстро эмулировать тысячи одновременных запросов, что идеально подходит для проверки масштабируемости системы в условиях реальной нагрузки.

- Простота использования: для запуска теста достаточно указать общее число запросов и число одновременных соединений.

Интеграция утилиты hey в процесс нагрузочного тестирования позволила получить глубокое понимание поведения сервера под различными уровнями нагрузки, что является ключевым моментом при оптимизации веб-приложений и обеспечении их стабильной работы даже в условиях пиковых нагрузок.[6]

Использование CPU и RAM— сколько вычислительных ресурсов потребляет сервер при обработке запросов. Для их исследование была использована утилита Process Explorer (и также встроеного мониторинга PowerShell для конкретного процесса Get-Process | Where-Object { $_.ProcessName -like "java" } | Select-Object Name, CPU, WS). Этот инструмент показывает использование CPU, RAM и другую системную информацию для выбранного процесса (таблица 4.3). [7]

Process Explorer — это продвинутая утилита от Microsoft (Sysinternals), которая позволяет детально отслеживать активные процессы и анализировать их использование системных ресурсов, включая CPU (центральный процессор) и RAM (оперативная память). [7]

Таблица 4.3 – использование CPU и RAM

	RAM		*CPU*	
Язык	**Низкая**	**Высокая**	**Низкая**	**Высокая**

Язык	**Низкая нагрузка (50 запросов)**	**Высокая нагрузка (5000 запросов)**	**Низкая нагрузка (50 запросов)**	**Высокая нагрузка (5000 запросов)**
Java	97,38 МБ	131,32 МБ	0,38%	6,82%
Node.js	65,79 МБ	110,73 МБ	0,38%	8,45%
Go	11,09 МБ	82,77 МБ	0,25%	3,80%

Go потребляет меньше всего памяти благодаря экономным горутинам. Node.js держится в пределах нормы, а Java требует больше памяти из-за тяжёлой модели потоков и работы JVM

Исходя из проведенных тестов, можно сделать вывод, что **Java** подходит для корпоративных приложений с комплексной логикой, но проигрывает по потреблению ресурсов и латентности при очень высокой нагрузке. **Node.js** отличен для I/O-зависимых задач и быстрой разработки. Лучшая производительность при средних нагрузках. **Go** идеален для высоконагруженных систем и требует меньше ресурсов. Выигрывает при большом количестве запросов.

Заключение

Исследование показало важность выбора подходящей модели обработки запросов для серверных веб-приложений в зависимости от специфики нагрузки. Асинхронные технологии, такие как Node.js и Go, продемонстрировали высокую эффективность при решении задач, связанных с интенсивным использованием ввода-вывода (I/O), обеспечивая стабильную производительность даже при высоких нагрузках. Эти технологии оказались более предпочтительными в сценариях с большим количеством одновременных запросов, где важно минимизировать задержки и ресурсоёмкость.

Многопоточные модели, такие как Java Threading и Python Threading, лучше подходят для вычислительно сложных задач, однако они сталкиваются с проблемами при высоких нагрузках из-за накладных расходов на создание и синхронизацию потоков. Это ограничивает их производительность в сценариях с большим количеством одновременных запросов.

Основные выводы исследования сводятся к следующим рекомендациям для разработки серверных приложений: для высоконагруженных I/O задач следует использовать асинхронные подходы, такие как Node.js и Go, в то время как для вычислительно интенсивных сценариев предпочтительнее использовать многопоточность. Также важно учитывать, что в некоторых случаях асинхронные и многопоточные модели могут дополнять друг друга, обеспечивая оптимальную производительность для различных типов нагрузки.

Особое внимание уделено управлению ресурсами сервера при масштабируемых нагрузках, что является ключевым аспектом для достижения высокой производительности и отказоустойчивости приложений. Использование правильных технологий и подходов в соответствии с типом задач позволит разработчикам более эффективно оптимизировать серверные приложения и повысить их безопасность.

Библиография

1. Опивалов С. А. Методы работы с потоками в языке Java // Международный журнал

гуманитарных и естественных наук. 2023. № 4 (79) Т. 3. С. 93-99.
2. Руководство по Node.js, часть 1: общие сведения и начало работы [Электронный ресурс]. URL: https://habr.com/ru/companies/ruvds/articles/422893/ (Дата обращения: 03.03.2025).
3. Опивалов С. А. Перспективы использования языка Котлина в программировании // Международный журнал гуманитарных и естественных наук. 2023. № 9 (84) Т. 1. С. 260-262.
4. Параллельное программирование в Go [Электронный ресурс]. URL: https://proglib.io/p/parallelnoe-programmirovanie-v-go-2021-05-23?ysclid=m7wag43tb6712881695 (Дата обращения: 03.03.2025).
5. Asynchronous Functions and the Node.js Event Loop [Электронный ресурс]. URL: https://translated.turbopages.org/proxy_u/en-ru.ru.137d7d27-67c89218-0c37f6b9-74722d776562/https/www.geeksforgeeks.org/asynchronous-functions-and-the-node-js-event-loop/ (Дата обращения: 04.03.2025).
6. Load Testing using Hey [Электронный ресурс]. URL: https://dev.to/saantoryuu/load-testing-using-hey-c84 (Дата обращения: 04.03.2025).
7. Process Explorer v17.06 [Электронный ресурс]. URL: https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer (Дата обращения: 02.03.2025).

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Тема особенно актуальна для облачных (cloud computing) и туманных (fog computing) глобально распределенных вычислительных систем, где нагрузка пользователей различных регионов мира характеризуется большими потоками данных параллельной обработки веб запросов (всемирная проблема big data), особенно на стороне сервера и баз данных.

Цель работы не сформулирована четко, хотя интуитивно понятно, что это повышение производительности веб приложений на стороне сервера в условиях повышенной нагрузки со стороны пользователей.

В работе приведен сравнительный обзор асинхронных и многопоточных подходов по критерию производительности серверных приложений. В качестве примеров асинхронных технологий рассмотрены Node.js, Python Asyncio, Go, Kotlin Coroutines, а многопоточных - Java Threading и Python Threading. Четко обозначены преимущества и недостатки каждой модели и приведены практические рекомендации их применения для разработки серверных приложений в зависимости от специфики нагрузки, что представляется как главная новизна работы.

Исследования доведены до программной реализации в виде нагрузочных тестов, что несомненно является достоинством данной работы. Приведены численные результаты среднего времени отклика и пропускной способности. Выявлено, что оба подхода справляются с увеличением пользовательской нагрузки. Незначительное отличие состоит в методах программирования, использования оперативной памяти и интеграции с другими программными системами.

Статья изложена грамотным техническим языком, понятным читательской аудитории в

данной предметной области. Структура оформления материала, введение, обзор технологий и подходов, методология исследований, реализация, заключение, библиография, соответствуют требованиям журнала. Таблицы приемлемого качества по критерию читабельности.

Критических замечаний не обнаружено. Работа может быть опубликована и соответствует тематике журнала "Программные системы и вычислительные методы".

Для улучшения качества работы и интереса читательской аудитории можно учесть следующие рекомендации:
1. Библиография содержит недостаточное количество источников 7, из них 5 это ссылки на Интернет ресурсы. Рекомендуется добавить именно научные публикации в журналах, желательно общедоступных.
2. В библиографии №1 грамматическая ошибка.
3. Ссылка в библиографии №5 не открывается.
4. Букву "ё" в тексте статьи не принято использовать, она может некорректно отображаться на сайтах Интернет и в системах цитирования, следует ее заменить на букву "е".
5. Фразы "мы замеряли" и "нами была использована" абстрактные, лучше конкретно указать кто, а лучше ссылку на источник информации.
6. Таблица 4.3 - надо уточнить единицу измерения, мегабит (Мб) или мегабайт (МБ).
7. Аббревиатура I/O расшифрована в заключении, а не при первом упоминании, и вообще необязательно расшифровывать I/O, CPU, RAM, т.к. они общеприняты в данной предметной области.

Общий вывод: статья может быть принята к публикации в журнале «Программные системы и вычислительные методы» с незначительными доработками.