ORES

Научно-практический электронный журнал
Оригинальные исследования (ОРИС), *ISSN 2222-0402*
www.ores.su
original-research.ru

УДК 004.051

РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ КАК КЛЮЧ К ЭФФЕКТИВНОСТИ ВЫСОКОНАГРУЖЕННЫХ ФИНАНСОВЫХ СИСТЕМ

Болдарев Максим Андреевич,
Магистрант, Московский физико-технический институт, г. Москва
boldarev.ma@phystech.edu

Аннотация

В статье представлено комплексное исследование эффективности реактивных архитектурных решений в разработке финансово-технических веб-приложений. Проведен сравнительный анализ синхронного и реактивного подходов к обработке данных с использованием современных технологических стеков на базе Java и Spring Boot. Экспериментальным путем доказана высокая результативность реактивной парадигмы программирования при обработке интенсивных транзакционных потоков. Исследование демонстрирует кратное превосходство реактивной архитектуры по пропускной способности и потреблению вычислительных ресурсов относительно традиционных синхронных решений. Представленные методологические подходы и архитектурные паттерны могут быть эффективно имплементированы в финансовых информационных системах для повышения их производительности и отказоустойчивости.

Ключевые слова: реактивное программирование, Асинхронная обработка, Финансовые веб-приложения, Spring Web Flux, Производительность.

REACTIVE PROGRAMMING AS A KEY TO EFFICIENCY IN HIGH-LOAD FINANCIAL SYSTEMS

Boldarev Maxim Andreevich,
Moscow Institute of Physics and Technology, MIPT, Phystech
boldarev.ma@phystech.edu

ABSTRACT

The article presents a comprehensive study of the effectiveness of reactive architectural solutions in the development of financial technology web applications. A comparative analysis of synchronous and reactive approaches to data processing was conducted using modern technological stacks based on Java and Spring Boot. The high performance of the reactive programming paradigm in processing intensive transaction flows was empirically proven. The study demonstrates multiple advantages of the reactive architecture in terms of throughput and computational resource consumption relative to traditional synchronous solutions. The presented methodological approaches and architectural patterns can be effectively implemented in financial information systems to improve their performance and fault tolerance.

Keywords: reactive programming, asynchronous processing, financial web applications, spring web flux, performance.

Введение:

Финансовые онлайн-платформы сегодня сталкиваются с высокими требованиями к эффективности и отказоустойчивости. По мере роста пользовательской базы и повышения ожиданий к качеству сервиса, создание оптимальных архитектурных решений становится критически важным аспектом финансовой разработки.

Среди ключевых проблем, с которыми сталкиваются финансово-технические приложения, особенно выделяются:

1. Увеличение пользовательской нагрузки, негативно влияющее на ключевые бизнес-показатели.
2. Избыточное потребление системных ресурсов отдельными компонентами приложения.
3. Постепенное снижение производительности системы в целом при масштабировании.

Эти факторы создают необходимость пересмотра традиционных подходов к архитектуре финансовых систем и поиска более эффективных решений для обеспечения стабильной работы в условиях растущих нагрузок и высоких стандартов обслуживания.

Разработка стратегий оптимизации становится не просто технической задачей, а бизнес-необходимостью для сохранения конкурентоспособности финансовых сервисов в современных условиях [1].

В современных финансовых приложениях рост числа транзакций в секунду часто вызывает серьезные проблемы с производительностью. Особенно заметны тайм-ауты в серверной части, связанные с частыми вызовами сборщика мусора в Java-подобных языках, что негативно влияет на время отклика сервисов. Стандартные методы оптимизации, такие как профилирование кода или настройка баз данных, становятся малоэффективными при достижении предельных значений использования ресурсов.

Повышенное потребление аппаратных ресурсов требует дополнительного масштабирования инфраструктуры хранения данных и реорганизации архитектуры приложений, что неизбежно ведет к усложнению системы и снижению ее надежности.

Корень проблемы кроется в использовании синхронных архитектурных решений. Большинство серверных частей финансовых систем разрабатываются на Java с применением Spring Boot, что позволяет быстро создавать функциональные веб-сервисы с широкими возможностями интеграции. Однако эта комбинация технологий обычно опирается на Apache Tomcat, который имеет существенные ограничения по производительности.

Apache Tomcat, будучи контейнером сервлетов с открытым кодом, по умолчанию ограничен 200 одновременными потоками для обработки запросов. При оптимальных условиях он способен обрабатывать до 1000 запросов в секунду, если время ответа не превышает 200 мс. Попытки увеличить количество потоков приводят к экспоненциальному росту потребления ресурсов и последующей деградации всей системы [2].

Веб-приложения сталкиваются с серьезными вызовами производительности при обработке до 4000 запросов в секунду на отдельный микросервис. Постоянный рост этих показателей вынуждает компании увеличивать количество физических и виртуальных серверов, добавлять ресурсы и принимать другие затратные меры масштабирования.

Традиционные синхронные архитектуры на базе Java и Spring Boot с Apache Tomcat демонстрируют существенные ограничения при высоких нагрузках. Использование многопоточного программирования с синхронизированными блоками приводит к экспоненциальному росту потребления физической памяти и быстрому заполнению non-heap секции JVM. Параллельно наблюдается перегрузка реляционных баз данных из-за лавинообразного увеличения запросов и ограниченных возможностей кэширования.

В качестве альтернативного решения рассматривается парадигма реактивного программирования, реализуемая через технологии Spring Web Flux, Reactive Streams и Reactive Data. Эти инструменты позволяют создавать системы, ориентированные на асинхронную обработку событий, включая взаимодействие с базами данных и внешними сервисами [3].

Ключевым элементом оптимизации выступает механизм Reactive Streams, обеспечивающий асинхронный поток данных с неблокирующим обратным давлением, схему которого можно наблюдать на рисунке 1. Этот подход позволяет эффективно согласовывать скорость обработки между источником и потребителем данных, предотвращая перегрузку системы при интенсивном потоке информации [4].

Реактивная архитектура предлагает более эффективное использование аппаратных ресурсов и повышенную пропускную способность, что особенно актуально для высоконагруженных финансовых приложений с растущими требованиями к производительности.

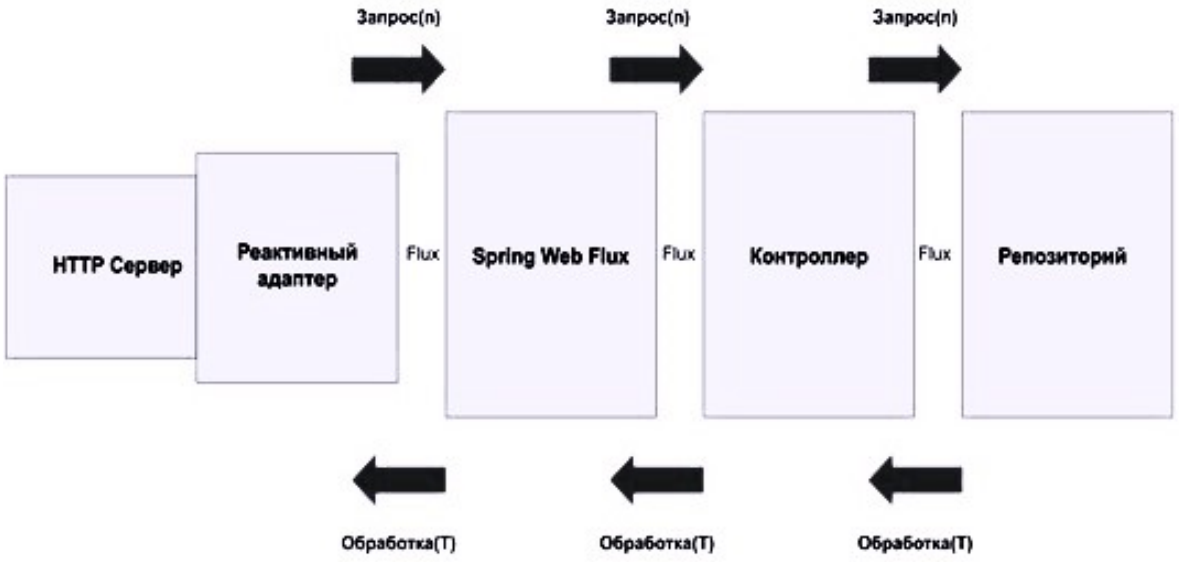

Рис. 1. Смеха неблокирующего обратного давления

Ожидания от Reactive Streams:

Потоковая обработка неограниченных данных - возможность работы с непрерывными потоками транзакций без блокировки системы.

Неблокирующее взаимодействие компонентов - асинхронный обмен данными между модулями системы повышает общую отзывчивость.

Интеллектуальное управление нагрузкой - механизм обратного давления автоматически регулирует поток данных, предотвращая перегрузки.

Важно отметить, что весь потенциал использования реактивного программирования раскрывается только при высокой интенсивности обработки запросов. При низкой нагрузке разницы при сравнительном анализе с синхронной обработкой наблюдаться не будет.

Цель исследования:

Разработка и оценка эффективности реактивной архитектуры для финансовых веб-приложений с целью повышения производительности, масштабируемости и эффективности использования вычислительных ресурсов в условиях высоких нагрузок.

Материалы и методы исследования:

Для проведения сравнительной оценки реактивного и синхронного подходов в разработке финансово-технических веб-сервисов были реализованы прототипы приложений, разработанные с помощью IntelliJ IDEA с использованием языка программирования Java, фреймворка Spring Boot, SQL и No SQL баз данных.

Прототип реактивного подхода включает использование технологии Web Flux для взаимодействия с внешними API, которая используется для реактивной обработки запросов в режиме Reactive Stream с использованием Mono и Flux, представляющие абстракцию для получения данных по модели «издатель-подписчик». Это означает, что при получении сигнатуры запроса параллельно формируются данные для ответа в одном потоке.

Система кэширования и хранения данных реализована на базе Reactive MongoDB Repository с применением специализированного драйвера MongoDB Reactive Streams, что обеспечивает неблокирующий доступ к информации.

В качестве источника внешних финансовых данных выступает официальный информационный ресурс Центрального банка РФ, откуда в режиме реального времени извлекаются актуальные котировки валютных пар для последующей обработки.

Аналитический компонент представлен интегрированным модулем прогнозирования на основе алгоритма ARIMA (AutoRegressive Integrated Moving Average), специализирующимся на анализе временных рядов финансовых показателей.

Такая комбинация технологий позволяет создать полностью реактивный конвейер обработки данных - от получения информации из внешних источников до её аналитической обработки и сохранения, с минимальными задержками и эффективным использованием вычислительных ресурсов.

Особую ценность представляет возможность асинхронного взаимодействия с базой данных, что исключает блокировки при интенсивном потоке финансовых транзакций и обеспечивает стабильную работу системы даже при пиковых нагрузках.

Прототип синхронного подхода включает аналогичную архитектуру взаимодействия компонентов с учетом замены реактивной технологии на Spring MVC, использующий стандартный HTTP-клиент и JDBC/JPA с PostgresQL в качестве синхронной SQL базы данных.

В рамках комплексного исследования эффективности разработанного решения был применен многоуровневый подход к сбору и анализу метрик производительности. В качестве основного инструмента нагрузочного тестирования использовался Apache JMeter, обеспечивающий моделирование различных сценариев пользовательской активности и сбор первичных данных о поведении системы под нагрузкой.

Для обеспечения всестороннего анализа производительности была развернута интегрированная система мониторинга на базе Prometheus и Grafana, позволяющая визуализировать и сопоставлять ключевые показатели эффективности, включая латентность обработки запросов, пропускную способность системы и параметры утилизации вычислительных ресурсов.

Сбор специализированных бизнес-метрик осуществлялся посредством JMeter с интегрированным Prometheus Backend Listener, что обеспечило непрерывную передачу данных в хранилище временных рядов Prometheus для последующего анализа и корреляции с техническими показателями.

Для получения детализированных метрик виртуальной машины Java были задействованы Spring Boot Actuator в сочетании с библиотекой Micrometer, что позволило

отслеживать внутренние процессы JVM в режиме реального времени. Дополнительно, для комплексной оценки эффективности использования ресурсов на уровне контейнеризации применялся Node Exporter, обеспечивающий сбор и экспорт данных об утилизации ресурсов изолированных окружений приложения.

Данный многокомпонентный подход к мониторингу позволил получить исчерпывающую картину производительности системы и выявить корреляции между различными аспектами функционирования реактивной финансовой платформы [5].

Результаты и их обсуждение:

Проведенное исследование выявило существенные различия в производительности между асинхронным неблокирующим сервером Netty, лежащим в основе Spring WebFlux, и традиционным сервлет-контейнером Apache Tomcat, используемым в Spring MVC. Эмпирические данные демонстрируют, что при низкоинтенсивных нагрузках обе технологии показывают сопоставимые результаты по пропускной способности. Однако с прогрессивным увеличением интенсивности входящего трафика Netty демонстрирует значительное преимущество, превосходя Tomcat в 1,5 раза по ключевым метрикам производительности, что наглядно отражено на графических материалах исследования на рисунке 2.

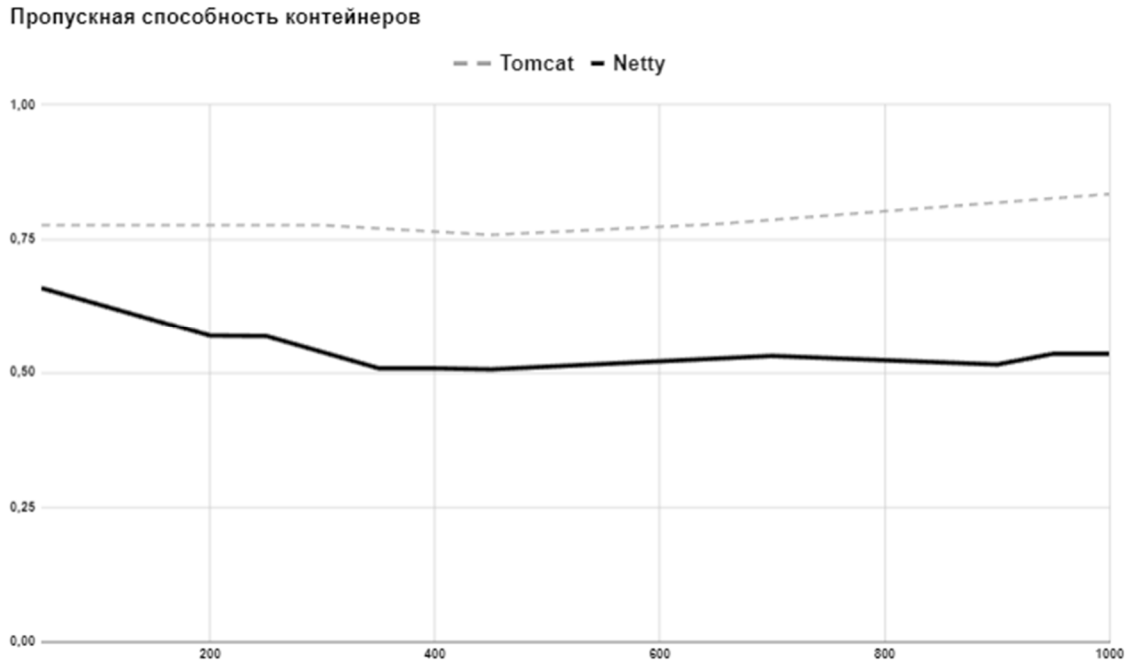

Рис. 2. Оценка пропускной способности реактивного и синхронного контейнеров

Фундаментальное различие между рассматриваемыми технологиями обусловлено принципиально разными подходами к обработке запросов. Классическая объектно-ориентированная модель, реализованная в Apache Tomcat, характеризуется жестким ограничением на количество одновременно обрабатываемых запросов (стандартный лимит составляет 200 параллельных потоков). В противовес этому, реактивная парадигма, воплощенная в Netty, функционирует на принципиально иной архитектуре, где вместо традиционного сервер-контейнера используется специализированный контроллер с единым основным потоком. При поступлении запроса система динамически создает новый поток, что позволяет масштабировать обработку практически без программных ограничений – единственным лимитирующим фактором становятся физические возможности процессора серверного оборудования.

Для достижения максимальной объективности в оценке преимуществ реактивного подхода исследование проводилось с использованием полностью реактивного технологического стека, включая компоненты доступа к данным. Это обеспечило целостность архитектурного решения и позволило в полной мере реализовать потенциал неблокирующей модели обработки запросов на всех уровнях приложения.

Например, анализ взаимодействия Java-приложений с системами управления базами данных выявляет фундаментальное ограничение: стандартные механизмы доступа к данным функционируют в блокирующем режиме, не поддерживая неблокирующий ввод/вывод. Это создает существенное препятствие для полноценной реализации преимуществ реактивной архитектуры, поскольку при отправке запроса в СУБД вычислительный поток вынужден приостанавливать выполнение до получения результата операции, тем самым удерживая дополнительные вычислительные ресурсы.

Исследование инфраструктуры современных финансовых приложений позволило идентифицировать три критические точки блокировки, потенциально снижающие эффективность асинхронной обработки:

Интеграционные интерфейсы с внешними API, используемые для агрегации финансовых данных.

Кэширующие структуры и репозитории, обеспечивающие промежуточное хранение информации.

Персистентные хранилища (СУБД), предназначенные для долговременного сохранения обработанных данных.

Для нивелирования негативного влияния указанных блокирующих компонентов предлагается комплексный архитектурный подход, основанный на принципах реактивного программирования. Взаимодействие с внешними API реорганизуется таким образом, что потоки выполняют исключительно идемпотентные операции записи по уникальным ключам с имплементацией механизмов защиты от конкурентной перезаписи. Данная методология исключает необходимость применения потокобезопасных коллекций и формирования очередей на запись, что существенно повышает пропускную способность системы.

Для обеспечения актуальности данных реализуется асинхронный механизм периодического опроса внешних API с предустановленными интервалами. Получаемая информация подвергается внутрипроцессной обработке и направляется в распределенный кэш и постоянное хранилище без блокировки основного потока обработки клиентских запросов. При инициализации приложения осуществляется предварительное наполнение кэша релевантными данными из базы, что минимизирует латентность при обслуживании начальных пользовательских запросов. При запуске приложения база данных заполняет кэш последними обработанными данными.

Предложенная методология позволяет радикально сократить временные и вычислительные затраты на обработку клиентских запросов, ограничивая их исключительно интервалом между моментом получения запроса и формированием ответа, полностью элиминируя из этого процесса временные издержки на взаимодействие с внешними API и системами хранения данных. Эмпирические исследования демонстрируют, что имплементация данного подхода обеспечивает значительное повышение производительности финансовых приложений при высоких нагрузках, сохраняя стабильность функционирования даже в условиях пиковых всплесков пользовательской активности.

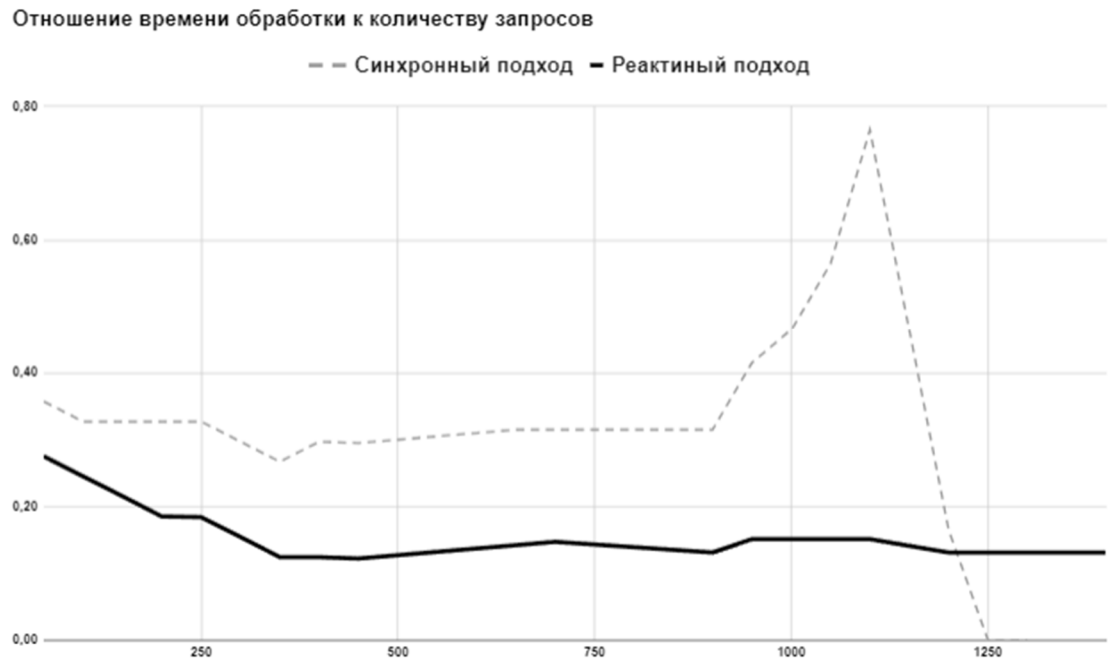

Рис. 3. Оценка времени обработки запроса в миллисекундах относительно количества запросов

Экспериментальные данные демонстрируют существенное превосходство реактивной парадигмы над традиционными синхронными методами обработки данных в контексте финансовых веб-приложений. Как иллюстрирует рисунок 3, пороговое значение пропускной способности для синхронной архитектуры составляет приблизительно 900 запросов в секунду, после чего наблюдается критическая деградация производительности, приводящая к генерации ошибок при обработке входящих запросов.

Контрастирующие показатели демонстрирует реактивная реализация, обеспечивающая стабильную обработку более 16 000 запросов в секунду при утилизации вычислительных ресурсов на уровне 80% в двухъядерной конфигурации. Данный результат свидетельствует о 17-кратном превосходстве в пропускной способности по сравнению с синхронным аналогом. Примечательно, что время обработки запросов в реактивной системе сохраняет стабильность даже при экспоненциальном увеличении нагрузки, в то время как синхронная модель демонстрирует прогрессирующее увеличение времени отклика.

Анализ утилизации вычислительных ресурсов, представленный на рисунке 4, выявляет еще более значительную дифференциацию: синхронный подход потребляет 7,5 милликора процессорного времени на обработку единичного запроса, тогда как реактивная архитектура требует лишь 0,015 милликора, что соответствует 500-кратному повышению эффективности использования CPU. Данная метрика была получена при стандартизированном сценарии нагрузочного тестирования с интенсивностью до 1000 запросов в секунду.

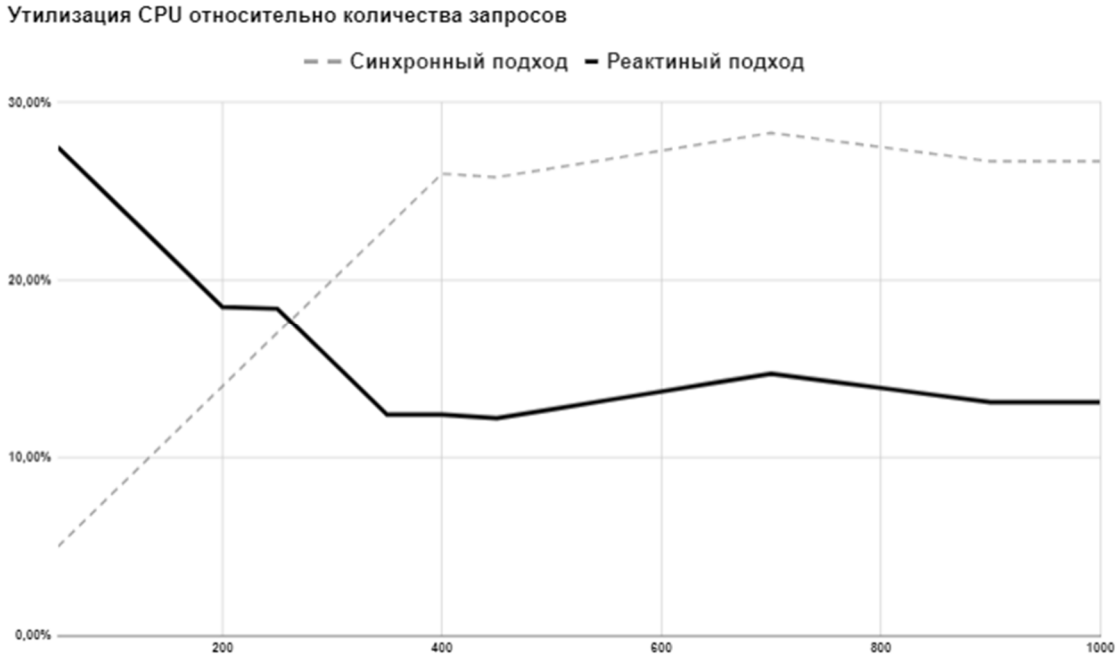

Рис. 4. Оценка потребления процессорного времени относительно количества запросов

Дополнительным преимуществом реактивного подхода является четырехкратное снижение потребления оперативной памяти, что коррелирует с существенным сокращением количества активных потоков в виртуальной машине Java. Данный фактор имеет особую значимость при развертывании приложений в контейнеризированных средах с ограниченными ресурсами.

Заключение:

Полученные эмпирические данные позволяют сделать обоснованный вывод о превосходстве реактивной парадигмы для имплементации современных финансово-технических веб-приложений. Реактивная архитектура обеспечивает значительно более высокую производительность и сниженную латентность при обработке масштабных транзакционных потоков, что полностью соответствует актуальным требованиям финансового сектора к отказоустойчивости и масштабируемости информационных систем.

Фундаментальным преимуществом реактивной методологии является её имманентная способность к обработке асинхронных событийных потоков с минимальной латентностью. Данная характеристика приобретает критическое значение в экосистеме финансовых приложений, функционирующих в условиях высоко волатильных рынков. Эмпирические наблюдения демонстрируют, что имплементация реактивных паттернов обеспечивает практически мгновенную нотификацию конечных пользователей о статусных изменениях транзакционных процессов и флуктуациях рыночных индикаторов, что существенно интенсифицирует информационную ценность предоставляемых сервисов.

Компаративный анализ вычислительных требований реактивных и императивных архитектур выявляет значительную дивергенцию в паттернах потребления серверных ресурсов. Реактивные системы демонстрируют существенно более эффективную утилизацию процессорных мощностей и оперативной памяти благодаря неблокирующим механизмам обработки данных и оптимизированному управлению потоками. Оценка данного феномена указывает на потенциал редукции инфраструктурных затрат, что трансформируется в конкурентное преимущество для финансовых организаций в

условиях интенсификации рыночной конкуренции и оптимизации операционных издержек.

Переход к реактивным архитектурным решениям в сфере разработки финансовых веб-приложений представляет собой не только технологический императив, но и стратегическую инициативу, соответствующую современным тенденциям цифровой трансформации. Данный подход обеспечивает мультифакторную оптимизацию ключевых параметров информационных систем, включая производительность, масштабируемость, повышенную эффективность использования вычислительных ресурсов, что в совокупности формирует комплексное конкурентное преимущество для финансовых институтов в эпоху акселерированной цифровизации.

Список литературы:

1. Борсов А.Ю. Проблемы развития финтеха в России // Скиф. Вопросы студенческой науки. 2023. № 1 (77). С. 397–403.
2. Белодед Н.И., Юрьев А.А. Реактивное программирование в Java с использованием Spring WebFlux // Взаимодействие науки и общества – путь к модернизации: сборник статей. 2023. С. 52.
3. Путнин В.И. Реализация веб сервиса с применением парадигмы реактивного программирования // Международный журнал прикладных наук и технологий «Integral». 2021. № 1. С. 342–346.
4. Маркин Е.И., Рябова К.М. Использование реактивного программирования при разработке мобильных приложений // Computational nanotechnology. 2016. № 2. С. 170–173.
5. Редкин П.А., Алёшкин А.С. Программный комплекс распределенного тестирования веб-приложений // International Journal of Open Information Technologies. 2024. Т. 12. № 4. С. 125–132.

References:

1. Borisov A.Y. Problems of fintech development in Russia // Skif. Questions of student science. 2023. No. 1 (77). pp. 397-403.
2. Beloded N.I., Yuryev A.A. Reactive programming in Java using Spring WebFlux // Interaction of science and society – the path to modernization: collection of articles. 2023. P. 52.
3. Putnin V.I. Implementation of a web service using the reactive programming paradigm // International Journal of Applied Sciences and Technologies "Integral". 2021. No. 1. pp. 342-346.
4. Markin E.I., Ryabova K.M. The use of reactive programming in the development of mobile applications // Computational nanotechnology. 2016. No. 2. pp. 170-173.
5. Redkin P.A., Alyoshkin A.S. Software package for distributed testing of web applications // International Journal of Open Information Technologies. 2024. Vol. 12. No. 4. pp. 125-132.