СРАВНИТЕЛЬНАЯ ОЦЕНКА БЛОКИРУЮЩЕЙ И НЕБЛОКИРУЮЩЕЙ МОДЕЛЕЙ ОБРАБОТКИ ЗАПРОСОВ В API-ШЛЮЗАХ МЕДИЦИНСКИХ ИНФОРМАЦИОННЫХ СИСТЕМ НА ОСНОВЕ ТЕОРИИ МАССОВОГО ОБСЛУЖИВАНИЯ

М.В. Ефимов, *аспирант*
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» (Россия, г. Санкт-Петербург)

DOI:10.24412/2500-1000-2026-5-1-313-324

Аннотация. *Актуальность исследования обусловлена ростом нагрузки на API-шлюзы медицинских информационных систем при обмене структурированными медицинскими данными и необходимостью предварительной оценки их производительности на этапе проектирования. Цель работы заключается в разработке и экспериментальной проверке методики сравнительной оценки блокирующей и неблокирующей моделей обработки запросов в API-шлюзах МИС на основе аппарата теории массового обслуживания. В качестве аналитической основы использованы модели многоканального обслуживания для блокирующей реализации thread-per-request и событийной обработки для неблокирующей реализации. Экспериментальная проверка выполнена на функционально эквивалентных реализациях API на базе Spring MVC и Spring WebFlux с использованием I/O-bound сценариев, имитирующих авторизацию врача, поиск пациента и передачу результатов лабораторных исследований. Полученные результаты показали, что в исследованной конфигурации неблокирующая реализация обеспечивает более устойчивое поведение при росте числа конкурентных соединений: пропускная способность увеличилась с 1150 до 2850 запросов/с, среднее время отклика снизилось с 740 до 320 мс, а потребление оперативной памяти при 2000 одновременных соединениях уменьшилось с 2,4 до 0,6 ГБ. Показано, что предложенная методика может использоваться для предварительного выбора архитектуры API-шлюза и оценки ресурсных требований до промышленной эксплуатации. Область применимости выводов ограничивается преимущественно I/O-bound сценариями; CPU-bound задачи, криптографическая обработка и массовая передача DICOM-изображений требуют отдельного анализа.*

Ключевые слова: *медицинские информационные системы; высоконагруженные API; теория массового обслуживания; блокирующая модель; реактивное программирование; пропускная способность; отказоустойчивость; Event-loop; цифровая медицина; экспериментальная верификация.*

Современные распределённые системы всё чаще используют неблокирующие и реактивные модели обработки запросов для обеспечения высокой пропускной способности и снижения задержек при большом числе конкурентных соединений. Теоретические преимущества неблокирующего ввода-вывода достаточно хорошо известны, однако эмпирическая оценка его поведения в корпоративных системах с неоднородными профилями нагрузки по-прежнему требует дополнительного исследования [1].

Процессы цифровой трансформации отечественного здравоохранения предъявляют повышенные требования к производительности, доступности и устойчивости медицинских информационных систем (МИС). Развитие единой государственной информационной системы в сфере здравоохранения (ЕГИСЗ), региональных сегментов и сервисов электронного медицинского документооборота сопровождается ростом объёмов обмена структурированными медицинскими данными. С учётом требований действующих нормативных документов, включая Приказ Минздрава РФ от 24.12.2018 № 911н и Приказ Минздрава России от 07.09.2020 № 947н, программные компоненты МИС должны обеспечивать стабильный обмен данными между медицинскими организациями, региональными системами и федеральными сервисами.

Значительная часть нагрузки в таких системах приходится на программные интерфейсы и API-шлюзы, через которые выпол-

няются операции авторизации пользователей, поиска пациентов, передачи результатов исследований, маршрутизации структурированных электронных медицинских документов (СЭМД) и взаимодействия с внешними сервисами. В отличие от задач, связанных с массовой передачей диагностических изображений DICOM, которые относятся к отдельному классу высокообъёмных нагрузок, в данной работе основное внимание уделяется I/O-bound сценариям обмена структурированными медицинскими сообщениями. Для таких сценариев существенная доля времени обработки запроса связана не с вычислениями на центральном процессоре, а с ожиданием ответа базы данных, внешнего сервиса или сетевого взаимодействия.

При проектировании API-шлюзов МИС возникает задача выбора между классической блокирующей моделью обработки запросов, основанной на подходе thread-per-request, и неблокирующей событийной архитектурой. Блокирующая модель проста в реализации и широко используется в корпоративных приложениях, однако при резком росте числа одновременных обращений может приводить к увеличению потребления оперативной памяти и росту очередей ожидания. Неблокирующая модель, основанная на событийном цикле, позволяет обслуживать большое число соединений ограниченным числом потоков, но требует более строгого контроля блокирующих операций и иной организации прикладного кода.

Существующие подходы к выбору архитектуры высоконагруженных API в медицинских системах часто опираются на эмпирические оценки и особенности используемого технологического стека. При этом для обоснованного проектирования требуется воспроизводимая методика, позволяющая связать расчётные характеристики системы с результатами нагрузочного тестирования. В качестве такого инструмента может быть использован аппарат теории массового обслуживания (ТМО), позволяющий оценивать коэффициент загрузки, среднее время ожидания, длину очереди и предельные режимы работы API-шлюза.

Целью работы является разработка и экспериментальная проверка методики сравнительной оценки блокирующей и неблокирующей моделей обработки запросов в API-шлюзах медицинских информационных систем на основе аппарата теории массового обслуживания и нагрузочного тестирования. В качестве экспериментальной основы рассматриваются реализации API на базе Spring MVC и Spring WebFlux, а область применимости выводов ограничивается типовыми I/O-bound сценариями обмена структурированными медицинскими данными.

Постановка задачи

Оперативный доступ к структурированным медицинским данным и СЭМД предъявляет повышенные требования к производительности API-шлюзов МИС. При росте числа конкурентных обращений такие шлюзы могут становиться узким местом, поскольку через них выполняются операции маршрутизации, обращения к базам данных, взаимодействия с внешними сервисами и передачи медицинских сообщений. Для указанных операций характерен I/O-bound профиль нагрузки, при котором существенная часть времени обработки связана с ожиданием сетевого обмена или ответа внешней системы.

В качестве предмета анализа рассматривается зависимость эксплуатационных характеристик API-шлюза от выбранной модели обработки запросов. Сопоставляются две архитектуры: блокирующая модель thread-per-request и неблокирующая событийная модель. Сравнение выполняется по метрикам, отражающим как производительность, так и ресурсную эффективность: коэффициент загрузки, среднее время отклика, длина очереди, пропускная способность, потребление оперативной памяти и накладные расходы на переключение контекста.

Для обеспечения сопоставимости результатов блокирующая и неблокирующая реализации формализуются в терминах теории массового обслуживания, после чего расчётные оценки проверяются экспериментально на реализациях API на базе Spring MVC и Spring WebFlux. Область применимости исследования ограничена типовыми I/O-bound сценариями обмена структурированными медицинскими данными. CPU-bound операции, криптографическая обработка, массовая передача DICOM-изображений и сервисы с блокирующими внешними зависимостями требуют отдельного анализа.

Теоретические исследования

Анализ производительности API-шлюзов медицинских информационных систем целесообразно выполнять не только на уровне программной реализации, но и на уровне модели обслуживания входящего потока запросов. Для этого могут использоваться методы теории массового обслуживания, позволяющие описывать поступление заявок, время их обработки, образование очередей и достижение предельных режимов загрузки. Такой подход особенно актуален для API-шлюзов, выполняющих маршрутизацию запросов, обращение к базам данных, взаимодействие с внешними медицинскими сервисами и передачу структурированных сообщений.

В блокирующей модели обработки запросов, соответствующей подходу thread-per-request, каждое входящее обращение связывается с отдельным потоком исполнения на всё время обработки. Если выполнение запроса включает ожидание ответа базы данных, лабораторной информационной системы или внешнего сервиса, соответствующий поток остаётся занятым до завершения операции. С точки зрения теории массового обслуживания такая архитектура может быть приближённо представлена как многоканальная система, где число каналов соответствует размеру пула потоков сервера приложений. При превышении интенсивности входящего потока над доступной пропускной способностью возникает очередь ожидания, увеличиваются задержки, а при исчерпании допустимых ресурсов возможны отказы в обслуживании.

Преимуществом блокирующей модели является простота программной реализации и предсказуемость последовательного выполнения операций. Как отмечается в работе А. Смирнова, синхронное взаимодействие через REST API обеспечивает понятную схему обмена: клиент отправляет запрос и ожидает ответа сервера, что упрощает управление ошибками и описание бизнес-процессов [3]. Однако при большом числе конкурентных соединений линейная зависимость между количеством активных запросов и числом выделенных потоков приводит к росту потребления оперативной памяти и накладных расходов на переключение контекста.

Неблокирующая модель обработки запросов использует иной принцип организации вычислений. Входящее соединение не требует выделения отдельного потока на всё время ожидания операции ввода-вывода. Вместо этого запрос регистрируется в событийном цикле, а продолжение обработки выполняется после готовности соответствующего I/O-события. Такой подход лежит в основе реактивных стеков, включая Netty и Spring WebFlux. Его преимущество проявляется прежде всего в I/O-bound сценариях, где значительная часть времени обработки связана с ожиданием ответа базы данных, сетевого взаимодействия или внешнего сервиса. В то же время наличие блокирующих операций внутри событийного контура может нивелировать преимущества неблокирующей архитектуры и приводить к росту задержек для других запросов.

Сравнение блокирующих и неблокирующих API рассматривается в ряде работ. В публикации A. Desenvolvedor [4] на примерах Python и Java показаны различия между синхронной и асинхронной обработкой, включая влияние ожидания I/O-операций на использование потоков исполнения. В работе da Silva A.F.M. и соавторов [7] выполнено экспериментальное сравнение блокирующих и неблокирующих моделей в REST API, где показано снижение времени отклика при переходе к реактивным паттернам в сценариях с высокой долей операций ввода-вывода. Эти результаты согласуются с предположением о целесообразности применения событийных архитектур для API-шлюзов МИС, обрабатывающих большое число конкурентных запросов к внешним сервисам и хранилищам данных.

Для построения аналитической модели важны также исследования, рассматривающие ограничения памяти и очередей в системах с высокой конкуренцией за ресурсы. В работах C. Nie и соавторов [5] аппарат теории массового обслуживания применяется для анализа устойчивости систем с ограниченным объёмом памяти промежуточного состояния [9, 10]. Несмотря на то, что исходный контекст связан с инференсом больших языковых моделей и управлением KV-кэшем, общий принцип моделирования систем, в которых производительность определяется не только интенсивностью входящего потока, но и объёмом сохраняемого состояния, может быть

использован при анализе API-шлюзов МИС. В частности, это относится к очередям необработанных сообщений, буферам сетевого обмена, кэшам нормативно-справочной информации и промежуточным структурам при валидации медицинских документов.

Вопросы параллельной обработки и управления внутренними очередями также важны для высоконагруженных API. В публикации J. Wang и соавторов [6] рассматривается формально верифицированный механизм work stealing для параллельной обработки. Хотя данный подход не является прямым аналогом событийного цикла WebFlux, он демонстрирует значимость корректной организации очередей задач и распределения нагрузки между исполнителями. Для API-шлюзов МИС это особенно важно в случаях, когда помимо сетевого обмена присутствуют дополнительные операции обработки сообщений, сериализации, валидации и маршрутизации.

Отдельное значение имеют механизмы ограничения нагрузки и защиты от перегрузки. T. Kalyanasundaram и соавторы [8] рассматривают подход к распределённому ограничению скорости запросов на уровне API-шлюзов. Rate limiting, backpressure и балансировка нагрузки позволяют предотвратить неконтролируемый рост очередей и снизить риск каскадной деградации сервисов. Эти механизмы не заменяют выбор модели обработки запросов, но должны рассматриваться совместно с ним при проектировании отказоустойчивых медицинских API.

Таким образом, теоретический анализ показывает, что выбор между блокирующей и неблокирующей моделями должен определяться характером нагрузки. Для CPU-bound задач, где основная доля времени расходуется на вычисления, преимущество неблокирующей модели может быть ограниченным. Напротив, для I/O-bound сценариев, типичных для API-шлюзов МИС при обращении к базам данных, внешним сервисам и системам обмена структурированными медицинскими сообщениями, событийная модель позволяет уменьшить зависимость между числом конкурентных соединений и числом потоков исполнения. Это создаёт предпосылки для более эффективного использования памяти и процессорного времени. Для количественной оценки указанных эффектов далее вводится математическая модель на основе аппарата теории массового обслуживания.

Математическая модель

Для количественной оценки производительности API-шлюза МИС введём модель входящего потока запросов и процесса их обслуживания. В рамках исследования рассматриваются преимущественно I/O-bound сценарии, в которых значительная часть времени обработки связана с ожиданием ответа базы данных, внешнего сервиса или сетевого взаимодействия. Блокирующая реализация API представляется как многоканальная система массового обслуживания, где число каналов соответствует размеру пула потоков сервера приложений. Неблокирующая реализация рассматривается как событийная модель обработки, в которой ограниченное число event-loop потоков обслуживает множество соединений без выделения отдельного потока на весь период ожидания I/O-операции.

Структурная схема прохождения запроса через блокирующий и неблокирующий API-шлюз представлена на рисунке 1.

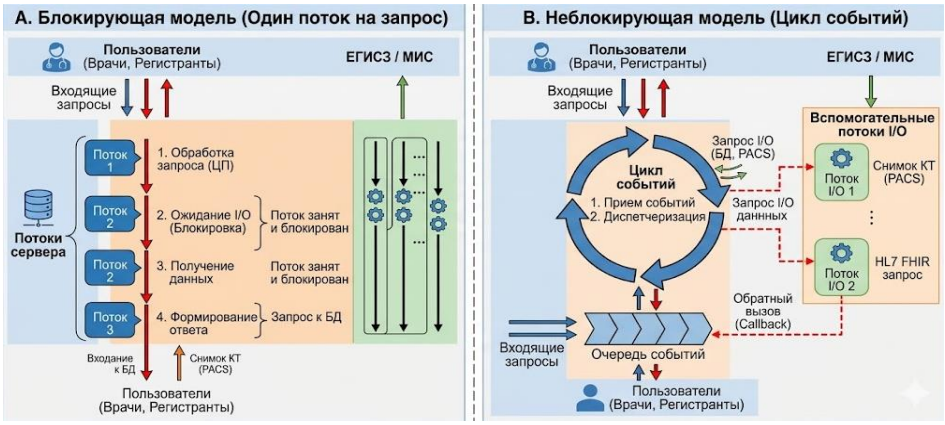

Рис. 1. Структурная модель процессов обработки запросов в блокирующем и неблокирующем API-шлюзах МИС

В блокирующей модели входящий запрос удерживает поток исполнения до завершения операции. При превышении числа одновременно обрабатываемых запросов над размером пула потоков формируется очередь ожидания или происходит отклонение новых соединений. В неблокирующей модели запрос регистрируется в событийном цикле, а продолжение обработки выполняется после готовности соответствующего события ввода-вывода. Это позволяет уменьшить число потоков исполнения, необходимых для обслуживания большого количества конкурентных соединений, однако требует исключения длительных блокирующих операций из event-loop.

Интенсивность входящего потока запросов определяется как:

$$\lambda = \frac{N}{T} \tag{1}$$

где N — число запросов, поступивших на вход API-шлюза за интервал наблюдения T; λ — средняя интенсивность поступления запросов, запросов/с.

Поскольку входящий поток медицинских сообщений неоднороден, среднее время обслуживания одного запроса определяется с учётом распределения типов операций:

$$E[S] = \sum_{i=1}^{k} p_i t_i \tag{2}$$

где p_i – вероятность поступления запроса i-го типа; t_i – среднее время обслуживания запроса данного типа; k – число типов запросов.

Интенсивность обслуживания одного канала равна:

$$\mu = \frac{1}{E[S]} \tag{3}$$

Для блокирующей модели, представляемой как система M/M/n, коэффициент загрузки вычисляется по формуле:

$$p_b = \frac{\gamma}{n\mu} \tag{4}$$

где n – число потоков в пуле сервера приложений; μ – интенсивность обслуживания одним потоком. При ρb→1 система приближается к режиму насыщения, что приводит к росту времени ожидания и длины очереди.

Вероятность ожидания в очереди для блокирующей модели оценивается с использованием формулы Эрланга C:

$$P_w = \frac{\frac{a^n}{n!} \cdot \frac{n}{n - a}}{\sum_{j=0}^{n-1} \frac{a^j}{j!} + \frac{a^n}{n!} \cdot \frac{n}{n - a}} \tag{5}$$

Показатель P_w характеризует вероятность того, что поступивший запрос не будет немедленно принят в обработку и попадёт в очередь. Среднее время ожидания в очереди для блокирующей модели определяется как:

$$W_{q,b} = \frac{P_w}{n\mu - \gamma} \tag{6}$$

Для неблокирующей модели используется приближённое описание событийной обработки. В общем случае реализация на базе WebFlux/Netty содержит несколько event-loop потоков, поэтому такая система может рассматриваться как M/G/k, где k – число event-loop потоков. Для аналитической оценки одного событийного цикла применяется частный случай M/G/1, для которого среднее время ожидания в очереди задаётся формулой Поллачека-Хинчина:

$$W_{q,nb} = \frac{\gamma E[S^2]}{2(1-\rho_{nb})} \quad (7)$$

E[S2] – второй начальный момент времени обслуживания. Данный показатель учитывает влияние разброса времени обслуживания запросов. Чем выше дисперсия времени обработки, тем сильнее отдельные длительные операции влияют на очередь событий. Поэтому для неблокирующей модели принципиально важно исключать синхронные блокирующие вызовы из event-loop.

Средняя длина очереди определяется на основе закона Литтла:

$$L_q = \lambda W_q \quad (8)$$

где Wq принимает значение Wq,b для блокирующей модели и Wq,nb для неблокирующей модели. Показатель Lq характеризует среднее число запросов, находящихся в ожидании обработки, и используется для оценки требуемого объёма буферов и памяти под контекст запросов.

Для сопоставления расчётной и экспериментальной производительности вводится оценка пропускной способности:

$$C = min(\lambda, \mu n \eta) \quad (9)$$

где C – фактическая пропускная способность системы, запросов/с; η – коэффициент, учитывающий накладные расходы среды исполнения, включая переключение контекста, синхронизацию потоков, работу сборщика мусора и сетевого стека. Для неблокирующей модели в выражении может использоваться число event-loop потоков k вместо n, а значение η определяется особенностями реализации и конфигурации среды выполнения.

Потребление оперативной памяти в блокирующей модели может быть приближённо представлено как:

$$M_b = M_0 + nm_{stack} + L_q m_{req} \quad (10)$$

где M0 – базовое потребление памяти приложением; m_{stack} – объём памяти, резервируемый под стек одного потока; mreq – средний объём памяти, необходимый для хранения контекста одного ожидающего запроса.

Для неблокирующей модели оценка потребления памяти имеет вид:

$$M_{nb} = M_0 + km_{stack} + L_q m_{event} + M_{buff} \quad (11)$$

где k – число event-loop потоков; mevent – средний объём памяти, связанный с хранением события или контекста асинхронной обработки; M_{buff} – память, используемая сетевыми и пользовательскими буферами. В отличие от блокирующей модели, число потоков k обычно существенно меньше числа конкурентных соединений, что снижает вклад стеков потоков в общее потребление памяти. При этом в неблокирующих реализациях возрастает значение корректного управления буферами и ограничениями очередей.

Предложенная модель позволяет оценивать влияние интенсивности входящего потока,

среднего времени обслуживания, размера пула потоков и характеристик очередей на задержки, пропускную способность и потребление памяти API-шлюза. Следует учитывать, что модели M/M/n и M/G/1 используются как аналитические приближения. Реальные API-шлюзы могут включать несколько последовательных узлов обслуживания: HTTP-сервер, бизнес-логику, пул соединений к базе данных, внешние сервисы и механизмы сериализации. Поэтому полученные расчётные значения далее сопоставляются с результатами экспериментального нагрузочного тестирования.

Экспериментальные исследования

Для проверки предложенной математической модели был разработан испытательный стенд с двумя функционально эквивалентными реализациями API на языке Kotlin: блокирующей реализацией на базе Spring MVC и неблокирующей реализацией на базе Spring WebFlux. Обе реализации поддерживали одинаковый набор прикладных сценариев: авторизация врача, поиск пациента и передача результата лабораторного исследования в формате JSON. Данные сценарии были выбраны как типовые I/O-bound операции медицинского API, поскольку значительная часть времени их выполнения связана с ожиданием ответа базы данных, внешнего сервиса или сетевого взаимодействия.

Экспериментальный стенд был развёрнут в инфраструктуре Yandex Cloud и включал три изолированных компонента: виртуальную машину с тестируемым API, отдельный экземпляр СУБД PostgreSQL 16 и отдельную виртуальную машину для генерации нагрузки. Сервисы API размещались на ВМ с гарантированными ресурсами 8 vCPU и 16 ГБ RAM под управлением Ubuntu LTS. На данной ВМ последовательно разворачивались две реализации API на Kotlin/JVM 24 с использованием Spring Boot 3.5.4: Spring MVC и Spring WebFlux. СУБД PostgreSQL 16 использовалась обеими реализациями с единой схемой данных, таблицами и индексами. Ресурсы СУБД были выделены с запасом относительно исследуемого диапазона нагрузок, что позволило исключить её как доминирующий фактор деградации производительности. Генерация нагрузки выполнялась с отдельной ВМ 2 vCPU и 4 ГБ RAM с использованием Gatling; все компоненты стенда располагались в одном регионе облачной инфраструктуры.

Нагрузочные сценарии включали 80% коротких операций со средним временем выполнения 100 мс и 20% длинных операций со средним временем ожидания 1200 мс. Длинные операции моделировали ожидание внешнего I/O: в блокирующей реализации ожидание удерживало рабочий поток, тогда как в неблокирующей реализации моделировалось без блокировки event-loop. Перед началом измерений выполнялась фаза разогрева, не учитываемая при обработке результатов. Метрики собирались средствами Micrometer, Prometheus и Grafana. Контролировались пропускная способность, задержки отклика, частота ошибок, потребление RAM, загрузка CPU, число активных потоков, а также параметры подключений и латентности запросов к PostgreSQL. Предварительный анализ метрик СУБД показал отсутствие насыщения по CPU, памяти и числу соединений в выбранном диапазоне нагрузок. Сравнительные характеристики блокирующей и неблокирующей реализаций представлены в таблице 1

Таблица 1. Сравнительный анализ расчетных и экспериментальных характеристик моделей обработки запросов в МИС

Параметр сравнения	Spring MVC	Spring WebFlux
Тип модели	M/M/n	M/G/k
Коэффициент загрузки ρ при λ=1000 запросов/с	0,865	-
Среднее время отклика, мс	740	320
Потребление RAM при 2000 соединениях, ГБ	2,4	0,6
Пропускная способность, запросов/с	1150	2850
Доля затрат CPU на переключение контекста, %	24,5	4,2
Порог выраженной деградации, соединений	850–900	не выявлен до 3000
Предпочтительная область применения	сервисы с умеренной конкурентностью; CPU-bound операции при выделении отдельных worker-пулов	I/O-bound API-шлюзы, интеграционные сервисы, телемедицина

Согласованность модели с экспериментом проверялась на основе параметров смешанного профиля нагрузки. При доле коротких запросов 80% и длинных запросов 20% среднее время обслуживания одного смешанного запроса составило:

$$E[S] = 0{,}8 \cdot 0{,}1 + 0{,}2 \cdot 1{,}2 = 0{,}32 \text{ с}$$

Полученное значение соответствует среднему времени отклика неблокирующей реализации Spring WebFlux, равному 320 мс, что указывает на отсутствие выраженного накопления очереди в исследованном диапазоне нагрузки. Для блокирующей реализации при эффективном размере пула n=370 потоков расчётная пропускная способность составила:

$$C_{calc} = n \cdot \frac{1}{E[S]} = 370 \cdot 3{,}125 = 1156 \text{ запросов/с}$$

Экспериментально измеренная пропускная способность Spring MVC составила 1150 запросов/с, отклонение расчётного значения от экспериментального не превысило 0,6%. Коэффициент загрузки при λ=1000 запросов/с составил:

$$\rho = \frac{\gamma}{n\mu} = \frac{1000}{370 \cdot 3{,}125} = 0{,}865$$

Такое значение соответствует режиму, близкому к насыщению. Дополнительно по закону Литтла для блокирующей реализации:

$$L = C \cdot R = 1150 \cdot 0{,}740 = 851$$

Полученное значение соответствует диапазону 850–900 конкурентных соединений, в котором экспериментально наблюдалась выраженная деградация времени отклика Spring MVC. Для Spring WebFlux аналогичная оценка даёт:

$$L = 2850 \cdot 0{,}320 = 912,$$

что согласуется с отсутствием резкой деградации в исследованном диапазоне до 3000 конкурентных пользователей.

Данные таблице 1 показывают, что в исследованном профиле нагрузки неблокирующая реализация обеспечивает более высокую пропускную способность и меньшее потребление оперативной памяти. При 2000 одновременных соединениях реализация на Spring MVC потребляла 2,4 ГБ RAM, тогда как реализация на Spring WebFlux – 0,6 ГБ. Это связано с тем, что в блокирующей модели рост числа конкурентных запросов требует увеличения числа потоков исполнения либо приводит к накоплению очереди. В неблокирующей модели большое число соединений обслуживается ограниченным числом event-loop потоков, что снижает вклад стеков потоков и связанных с ними структур в общее потребление памяти.

На рисунке 2 представлена зависимость среднего времени отклика от числа конкурентных пользователей.

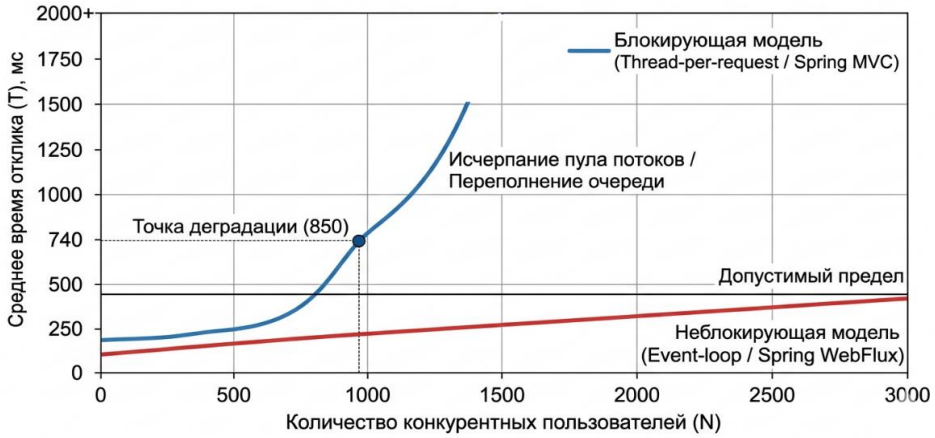

Рис. 2. Зависимость среднего времени отклика от количества конкурентных пользователей

Анализ зависимости показывает, что для блокирующей реализации после достижения диапазона 850-900 одновременных соединений наблюдается выраженный рост времени отклика. Данный эффект объясняется насыщением пула потоков и увеличением очереди ожидания. Неблокирующая реализация в исследованном диапазоне нагрузок не демонстрировала резкой деградации времени отклика и сохраняла среднее значение задержки около 320 мс при нагрузке до 3000 конкурентных пользователей. Следует отметить, что при дальнейшем росте интенсивности запросов неблокирующая модель также может перейти к режиму насыщения из-за ограничений event-loop потоков, сетевого стека, пулов соединений или внешних сервисов.

На рисунке 3 приведено сравнение пропускной способности реализаций.

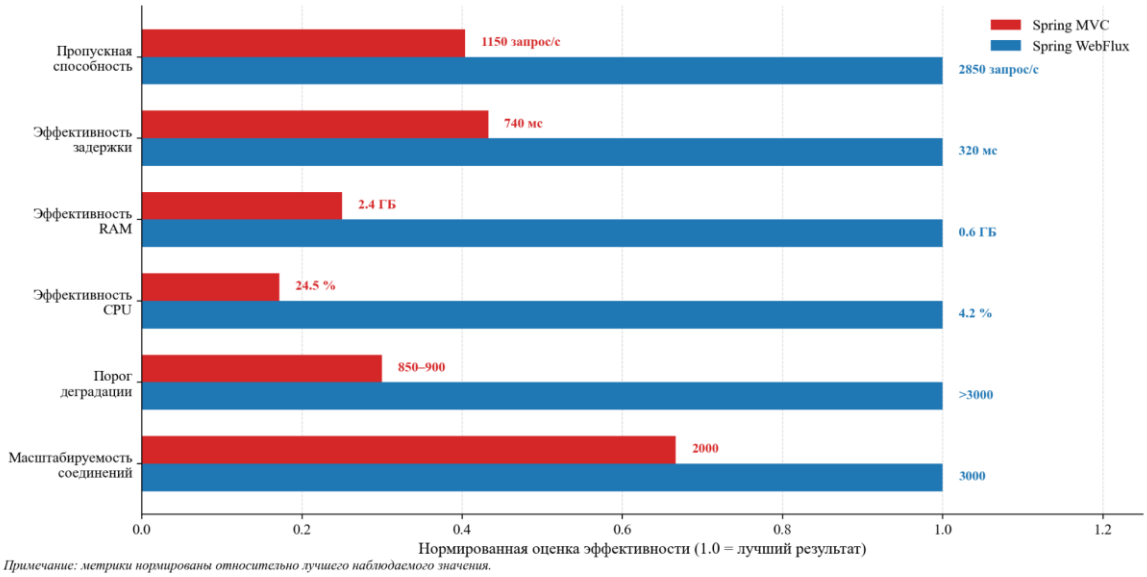

Рис. 3. Сравнительная характеристика пропускной способности систем при обмене СЭМД

График показывает, что блокирующая реализация достигает плато производительности раньше, после чего дальнейший рост числа соединений не приводит к пропорциональному увеличению числа успешно обработанных запросов. Для неблокирующей реализации в исследованном диапазоне наблюдается более устойчивый рост пропускной способности, достигающий 2850 запросов/с. Это подтверждает целесообразность применения событийной модели для I/O-bound сценариев медицинского обмена, где значительная часть времени запроса связана с ожиданием внешних операций.

На рисунке 4 показана динамика использования оперативной памяти.

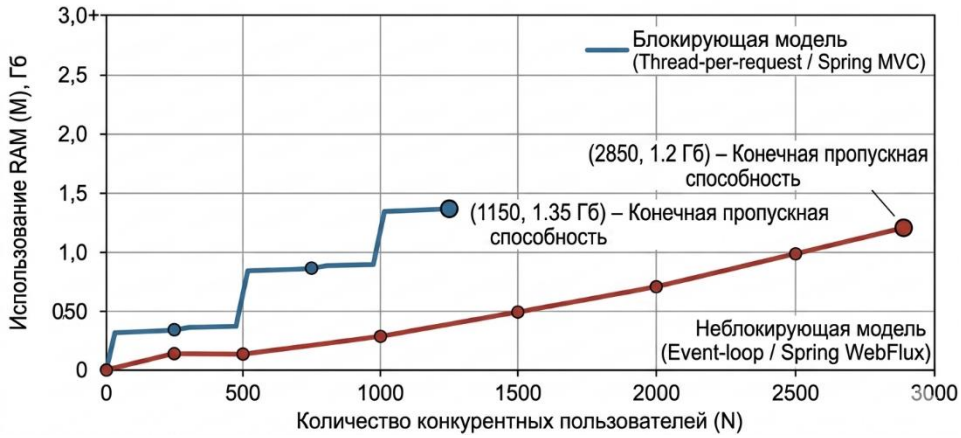

Рис. 4. Динамика использования оперативной памяти сервером приложений при росте нагрузки

В блокирующей реализации рост числа одновременных соединений сопровождается увеличением потребления памяти, что связано с резервированием стеков потоков и хранением контекстов запросов. В неблокирующей реализации рост потребления памяти выражен слабее, поскольку число event-loop потоков не увеличивается пропорционально числу соединений. При этом необходимо учитывать, что неблокирующие системы также используют память под сетевые буферы, очереди событий и структуры асинхронной обработки, поэтому итоговое потребление ресурсов зависит от конфигурации runtime и характера нагрузки.

На рисунке 5 представлено влияние переключений контекста на загрузку процессора.

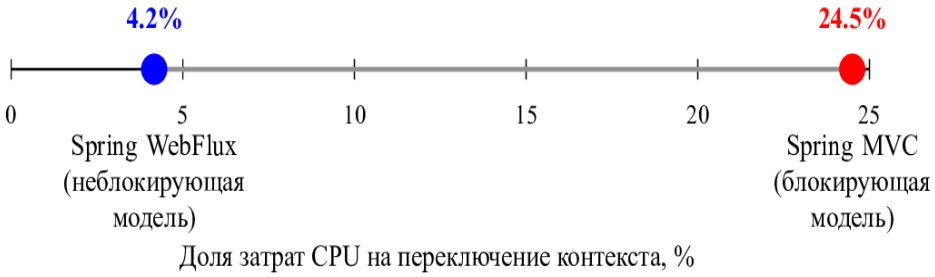

Рис. 5. Влияние количества переключений контекста на загрузку центрального процессора

Результаты показывают, что в блокирующей модели при росте числа потоков увеличивается доля процессорного времени, затрачиваемая на управление потоками и переключение контекста. В неблокирующей модели данный показатель ниже, поскольку обработка большого числа соединений выполняется ограниченным числом потоков. Это снижает накладные расходы и позволяет эффективнее использовать вычислительные ресурсы при I/O-bound нагрузке.

Таким образом, экспериментальные данные подтверждают применимость предложенной модели для предварительной оценки производительности API-шлюзов МИС. В рассматриваемых сценариях обмена структурированными медицинскими данными неблокирующая реализация обеспечила меньшее среднее время отклика, более высокую пропускную способность и снижение потребления оперативной памяти по сравнению с блокирующим подходом. Вместе с тем полученные выводы не следует переносить на все типы медицинских сервисов. Для CPU-bound задач, криптографической обработки, генерации отчётов, обработки DICOM-изображений и сервисов с блокирующими внешними зави-

симостями требуется отдельное моделирование и экспериментальная проверка.

Заключение

В работе выполнен сравнительный анализ блокирующей и неблокирующей моделей обработки запросов в API-шлюзах медицинских информационных систем. Для формализации поведения рассматриваемых архитектур использован аппарат теории массового обслуживания, позволяющий на этапе проектирования выполнять предварительную оценку коэффициента загрузки, области насыщения, пропускной способности и потребления оперативной памяти.

Экспериментальная проверка на реализациях Spring MVC и Spring WebFlux показала, что для исследованных I/O-bound сценариев обмена структурированными медицинскими данными неблокирующая модель обеспечивает более устойчивое поведение при росте числа конкурентных соединений. Блокирующая реализация демонстрировала выраженную деградацию времени отклика при 850-900 одновременных соединениях, тогда как неблокирующая не показывала резкой деградации при нагрузке до 3000 пользователей. Пропускная способность Spring WebFlux достигла 2850 запросов/с против 1150 запросов/с у Spring MVC, среднее время отклика снизилось с 740 до 320 мс, а потребление оперативной памяти API-процесса при 2000 соединениях – с 2,4 до 0,6 ГБ.

Согласованность расчётной модели с экспериментом подтверждена на профиле нагрузки 80/20: 80% коротких запросов по 100 мс и 20% длинных запросов по 1200 мс. Среднее базовое время обслуживания составило 320 мс. Для блокирующей реализации расчётная пропускная способность при эффективном размере пула 370 потоков составила 1156 запросов/с, что отличается от экспериментального значения менее чем на 1%. Оценка по закону Литтла дала значение 851 запроса в системе, что соответствует наблюдаемой области деградации при 850-900 соединениях.

Практически это означает, что для I/O-bound API МИС с высокой конкурентностью неблокирующая модель должна рассматриваться как предпочтительный вариант. При этом CPU-bound и блокирующие операции необходимо выносить из event-loop в отдельные worker-пулы или асинхронные задания, а окончательный выбор архитектуры подтверждать нагрузочным тестированием с контролем задержек, ошибок, памяти и пулов соединений. Полученные выводы не распространяются напрямую на криптографическую обработку, генерацию отчётов, массовую передачу DICOM-изображений и иные вычислительно интенсивные задачи.

Библиографический список

1. Zbarcea A., Tudose C., Boicea A. Extending the Migration from Asynchronous to Reactive Programming in Java: A Performance Analysis of Caching, CPU-Bound, and Blocking Scenarios // Applied Sciences. – 2026. – Vol. 16. № 1. – P. 90.

2. Wang H., Wei J., Yan T., Qiang L., Li H. Study and Optimization of Server Load Capacity in High Concurrency Scenarios // International Journal of Advanced Network, Monitoring and Controls. – 2025. – Vol. 10. – Pp. 62-81.

3. Smirnov A. Comparative analysis of performance and scalability of synchronous and asynchronous interactions in microservice architecture // Professional Bulletin: Information Technology and Security. – 2025. – № 3. – Pp. 15-20.

4. Spring Framework Documentation. Web on Reactive Stack // VMware, Inc. 2024. – [Электронный ресурс]. – Режим доступа: https://docs.spring.io/spring-framework/reference/web/webflux.html.

5. Nie C., Si N., Zhou Z. A Queueing-Theoretic Framework for Stability Analysis of LLM Inference with KV Cache Memory Constraints // ICLR (under review). – 2026. – Pp. 1-17.

6. Wang J., Trach B., Fu M., Behrens D., Schwender J., Liu Y., Lei J., Vafeiadis V., Härtig H., Chen H. BWoS: Formally Verified Block-based Work Stealing for Parallel Processing // Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI '23). – 2023. – Pp. 833-850.

7. da Silva A.F.M., da Rocha Balthazar G., Ferreira E.A., dos Santos E.S. Comparative analysis of blocking and non-blocking models in rest APIS // Cuadernos de Educación y Desarrollo. – 2025. – Vol. 17. № 8. – Art. e9049.

8. Kalyanasundaram T., Panchalingam K., Jegatheesan T., Wijayasiri A., Perera S. Load Balancer Filter-Based Approach To Enable Distributed API Rate Limiting // 37th Conference of Open Innovations Association (FRUCT). – 2025. – Pp. 75-85.

9. Gross D., Shortle J.F., Thompson J.M., Harris C.M. Fundamentals of Queueing Theory. 5th ed. – Hoboken: John Wiley & Sons, 2018. – 500 p.

10. Harchol-Balter M. Performance Modeling and Design of Computer Systems: Queueing Theory in Action. – Cambridge: Cambridge University Press, 2013. – 574 p.

A COMPARATIVE EVALUATION OF BLOCKING AND NON-BLOCKING REQUEST PROCESSING MODELS IN API GATEWAYS OF MEDICAL INFORMATION SYSTEMS BASED ON QUEUEING THEORY

M.V. Efimov, *Postgraduate Student*
Saint Petersburg Electrotechnical University "LETI"
(Russia, Saint Petersburg)

Abstract. *The relevance of this study is determined by the increasing load on API gateways of medical information systems caused by the exchange of structured medical data and by the need for preliminary performance assessment at the system design stage. The aim of the study is to develop and experimentally validate a methodology for comparative evaluation of blocking and non-blocking request processing models in MIS API gateways using queueing theory. As an analytical basis, a multichannel service model is used for the blocking thread-per-request implementation, while an event-driven processing model is used for the non-blocking implementation. Experimental validation was performed using functionally equivalent API implementations based on Spring MVC and Spring WebFlux under I/O-bound scenarios simulating physician authentication, patient search, and transmission of laboratory test results. The results show that, in the studied configuration, the non-blocking implementation provides more stable behavior as the number of concurrent connections increases: throughput increased from 1,150 to 2,850 requests/s, the average response time decreased from 740 to 320 ms, and RAM consumption under 2,000 simultaneous connections decreased from 2.4 to 0.6 GB. The proposed methodology can be used for preliminary selection of an API gateway architecture and estimation of resource requirements before production deployment. The applicability of the conclusions is primarily limited to I/O-bound scenarios; CPU-bound tasks, cryptographic processing, and bulk transfer of DICOM images require separate analysis.*

Keywords: *medical information systems; high-load APIs; queueing theory; blocking model; reactive programming; throughput; fault tolerance; event loop; digital medicine; experimental verification.*