УДК 004.42:004.5

# Оптимизация производительности микросервисной архитектуры в финтех-проектах

**С.Н. Болгов**✉

*Мурманский арктический университет, Мурманск, Россия*

*В статье рассматриваются методы и подходы к оптимизации производительности микросервисной архитектуры в финтех-проектах. Особое внимание уделяется выбору технологий и инструментов для эффективного взаимодействия сервисов, минимизации задержек, обеспечению масштабируемости и надежности. Анализируются ключевые аспекты, такие как использование REST и gRPC, асинхронных очередей сообщений и сервис-мешей (service mesh), а также балансировка нагрузки и динамическое масштабирование. Приводятся примеры практических решений для повышения производительности в условиях высоких требований к финансовым сервисам, а также рекомендации по мониторингу и тестированию.*

*Ключевые слова: микросервисная архитектура, финтех, производительность, масштабируемость, REST, gRPC.*

# Optimizing microservices architecture performance in fintech projects

**S.N. Bolgov**✉

*Murmansk Arctic University, Murmansk, Russia*

*The article explores methods and approaches to optimize microservices architecture performance in fintech projects. Special attention is given to selecting technologies and tools for efficient service interaction, minimizing latency, ensuring scalability and reliability. Key aspects such as the use of REST and gRPC, asynchronous message queues, service meshes, as well as load balancing and dynamic scaling are analyzed. Practical solutions for improving performance under high requirements for financial services are presented, along with recommendations for monitoring and testing.*

*Keywords: microservices architecture, fintech, performance, scalability, REST, gRPC.*

## Введение

Современный финансовый сектор предъявляет повышенные требования к производительности и надежности информационных систем. Финтех-компании оперируют огромными объемами данных, обрабатывают миллионы транзакций в реальном времени и обеспечивают соблюдение строгих регуляторных норм. В этих условиях выбор архитектурных решений становится особенно важным фактором, определяющим успешность цифровых платформ.

Исторически финансовые системы строились на монолитных архитектурах, обеспечивавших целостность данных и жесткий контроль бизнес-логики. Однако с ростом нагрузки и необходимостью быстрой адаптации к изменениям рынка традиционные подходы стали ограничивать масштабируемость и замедлять внедрение инноваций. В ответ на эти вызовы финтех-отрасль активно переходит на микросервисные архитектуры, предлагающие гибкость, модульность и независимое развертывание компонентов.

© Болгов С.Н., 2025

Несмотря на очевидные преимущества, микросервисные решения сопровождаются новыми проблемами, среди которых ключевую роль играет снижение производительности. Децентрализация приводит к увеличению расходов на сетевые взаимодействия, усложняет управление данными и требует эффективных механизмов оркестрации сервисов. В этой связи возникает вопрос: какие архитектурные решения позволяют оптимизировать производительность микросервисных систем в условиях высокой нагрузки и строгих требований финтех-индустрии?

Настоящее исследование направлено на анализ влияния различных архитектурных подходов на ключевые показатели производительности микросервисных платформ. В работе рассматриваются модели взаимодействия сервисов, методы управления состоянием, стратегии масштабирования и их влияние на вычислительную эффективность финтех-приложений.

## Основная часть. Архитектурные подходы в финтехе: переход от монолитных систем к микросервисам

Финансовые системы традиционно разрабатывались на основе монолитной архитектуры, при которой все компоненты приложения объединены в единую кодовую базу и функционируют в одном исполняемом окружении. Такой подход обеспечивал строгую консистентность данных, надежность транзакционных операций и централизованное управление бизнес-логикой. Однако с ростом объемов обрабатываемой информации и усложнением пользовательских сценариев монолитные решения начали сталкиваться с рядом ограничений. В первую очередь, это проблемы с масштабируемостью, так как увеличение нагрузки требует горизонтального или вертикального масштабирования всей системы, а не отдельных ее компонентов. Это приводит к избыточному потреблению ресурсов и снижению операционной эффективности [1].

Другой важный аспект – скорость развертывания и обновления функциональности. В условиях высококонкурентного рынка финтех-компании вынуждены оперативно внедрять новые продукты и сервисы. Однако внесение изменений в монолитное приложение требует полной перекомпиляции и развертывания всей системы, что увеличивает время вывода новых функций на рынок. Кроме того, масштабные изменения в кодовой базе повышают риск появления ошибок, усложняют тестирование и ухудшают управляемость программного продукта. Эти факторы способствовали переходу финансового сектора к микросервисной архитектуре, обеспечивающей гибкость, модульность и независимость компонентов (рис.) [2].

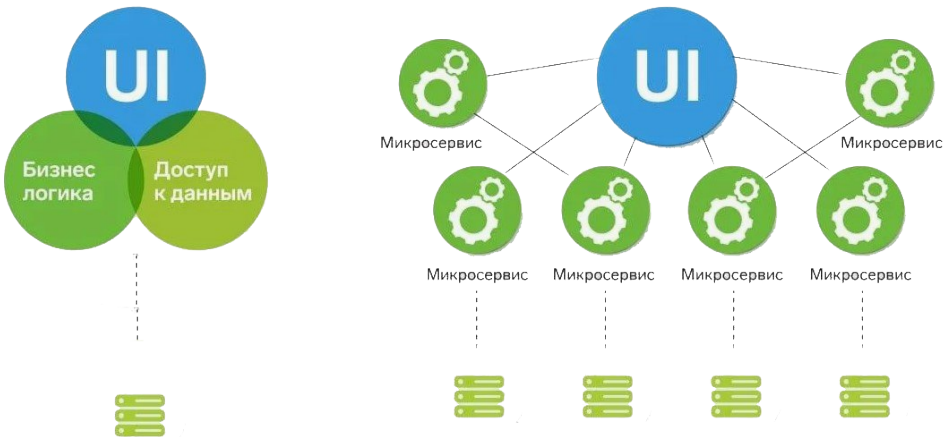


Рисунок. Разница между монолитной и микросервисной архитектурой

Микросервисный подход разделяет систему на автономные сервисы, каждый из которых выполняет одну бизнес-функцию и взаимодействует через стандартизированные интерфейсы. Это позволяет масштабировать только нужные компоненты, что важно для высоконагруженных финансовых сервисов, работающих в реальном времени. Например, платежный шлюз можно масштабировать отдельно от модуля аналитики, что эффективно распределяет вычислительные ресурсы.

Преимуществом микросервисов является независимая разработка и развертывание компонентов, ускоряющих внедрение новых функций. Однако это вызывает сложности в координации сервисов, управлении данными и мониторинге. В отличие от монолитных систем, где согласованность данных централизована, в микросервисах необходимо использовать распределенные механизмы синхронизации, такие как event sourcing, CQRS и Saga [3].

Использование микросервисов увеличивает сетевые расходы из-за вызовов через HTTP, gRPC или брокеры сообщений, что может увеличивать задержки и снижать производительность. Для финтех-приложений важно применять стратегии межсервисного взаимодействия, такие как API Gateway, сервисная шина и асинхронные очереди сообщений (Kafka, RabbitMQ). Таким образом, переход от монолита к микросервисам в финтехе обусловлен стремлением к гибкости, масштабируемости и ускоренному развертыванию новых сервисов, однако он требует внедрения дополнительных инструментов для обеспечения согласованности данных, мониторинга и управления распределенными компонентами.

## Методы оценки производительности микросервисов

Эффективность микросервисной архитектуры в финтех-приложениях зависит не только от гибкости и масштабируемости, но и от способности обеспечивать высокую производительность с минимальными задержками. В отличие от монолитных систем, где производительность оценивается в рамках одного процесса, микросервисные платформы требуют комплексного мониторинга, учитывающего сетевые взаимодействия, балансировку нагрузки и распределенные вычисления. Одним из ключевых методов оценки является метрика пропускной способности (throughput), показывающая количество обработанных запросов в единицу времени. Для финтех-приложений, например, в платежных системах, важно поддержание стабильного throughput, так как его снижение может привести к финансовым потерям. Пропускная способность анализируется с помощью нагрузочного тестирования, например, с использованием инструментов JMeter или Locust.

Другим важным показателем является время отклика (latency) – промежуток времени между отправкой запроса и получением ответа. В микросервисной архитектуре латентность может увеличиваться из-за сетевых взаимодействий и необходимости координации между множеством сервисов. Важно различать среднюю латентность и перцентильные значения (например, p95 и p99), которые отражают задержки в наиболее загруженных сценариях. Для мониторинга латентности используются инструменты распределенного трейсинга, такие как OpenTelemetry, Jaeger и Zipkin, позволяющие отслеживать цепочку вызовов внутри распределенной системы и выявлять проблемы [4].

Дополнительно оценивается уровень потребления ресурсов, включая загрузку процессора (CPU), использование оперативной памяти (RAM) и сетевые затраты. В микросервисных архитектурах нагрузка распределяется неравномерно, что требует анализа hotspots – узлов, испытывающих наибольшую нагрузку. Для балансировки используется динамическое масштабирование (autoscaling), при котором новые инстансы сервисов создаются автоматически в ответ на увеличение нагрузки. Однако

избыточное масштабирование может привести к перерасходу ресурсов, поэтому применяются алгоритмы адаптивного масштабирования, учитывающие исторические данные нагрузки и прогнозируемые пики.

Для комплексной оценки производительности микросервисов используются аппаратно-программные метрики, объединяющие показатели нагрузки с бизнес-метриками. Например, в финансовых системах анализируется не только техническая производительность, но и такие параметры, как скорость обработки транзакций, количество отклоненных платежей из-за тайм-аутов и соответствие требованиям регуляторов по обработке данных в реальном времени. Оценка производительности микросервисной архитектуры требует применения комплексного подхода, включающего анализ пропускной способности, задержек, использования ресурсов и распределенной нагрузки.

## Влияние архитектурных решений на производительность

Производительность микросервисной архитектуры в финтех-приложениях определяется не только используемыми технологиями, но и принципами проектирования распределенных систем. Архитектурные решения оказывают непосредственное влияние на задержки, пропускную способность и потребление ресурсов, что особенно важно для финансовых платформ, работающих в условиях высокой нагрузки и строгих требований к надежности [5].

Одним из ключевых факторов, влияющих на производительность, является способ межсервисного взаимодействия. Использование REST API с JSON-сериализацией увеличивает расходы на обработку запросов и передачу данных, что приводит к росту задержек. В высоконагруженных системах REST заменяется на gRPC, обеспечивающий бинарную сериализацию и поддержку потоковой передачи данных. Это снижает среднюю задержку вызовов что особенно важно для обработки транзакций и операций с биржевыми котировками. Синтетические тесты на производительность показали, что gRPC показывает время ответа в 7,077 секунды на 1000 запросов, в то время как REST в аналогичных условиях – 43,674 секунды [6]. Дополнительно оптимизировать взаимодействие сервисов позволяют асинхронные очереди сообщений (Kafka, RabbitMQ), уменьшающие нагрузку на сервисы за счет распределенной обработки данных и предотвращения перегрузок.

В микросервисной архитектуре производительность зависит от управления состоянием сервисов. Stateful-сервисы требуют постоянного доступа к базе данных, что снижает отказоустойчивость и увеличивает количество операций. В высоконагруженных системах предпочтительнее stateless-подход, при котором состояние хранится во внешних распределенных хранилищах, таких как Redis или Apache Ignite.

Оптимизация доступа к данным также важна. В традиционных реляционных базах данных блокировки ограничивают параллельность операций, тогда как NoSQL-хранилища, например Cassandra или MongoDB, обеспечивают горизонтальное масштабирование. Переход к NoSQL требует применения паттернов Event Sourcing и CQRS для разделения операций чтения и записи [7].

Использование сервис-мешей (Istio, Linkerd) позволяет управлять трафиком и балансировать нагрузку, что критично для финтех-платформ с большим числом одновременных соединений, где даже незначительное снижение задержек улучшает пользовательский опыт и стабильность системы.

Выбор архитектурных решений напрямую определяет производительность микросервисных систем, влияя на межсервисное взаимодействие, управление состоянием, работу с базами данных и балансировку нагрузки.

**Особенности тестирования производительности микросервисных систем**

Для объективной оценки производительности микросервисной архитектуры в финтех-приложениях необходимо практическое сравнение различных техник и инструментов для тестирования производительности систем [8, 9]. Это позволит не только выявить проблемы, но и оценить влияние различных архитектурных решений на ключевые метрики финтех-проектов (табл.).

Таблица

Техники и инструменты для тестирования производительности систем

| Инструмент | Преимущества | Недостатки | Применение |
|---|---|---|---|
| Apache JMeter | Поддержка различных протоколов, масштабируемость, поддержка распределенного тестирования | Высокие требования к ресурсам при тестировании больших нагрузок | Веб-приложения, API, базы данных, SOAP, REST |
| LoadRunner | Широкий спектр поддерживаемых протоколов, аналитика, интеграция с ALM | Сложность настройки, стоимость лицензии | Корпоративные системы, ERP, CRM, мобильные приложения |
| Gatling | Высокая производительность, открытый исходный код, интеграция с CI/CD | Менее развитый интерфейс, ограниченная поддержка протоколов | Веб-приложения, API, микро-сервисы |
| BlazeMeter | Облачная платформа, масштабируемость, интеграция с CI/CD | Зависимость от облака, высокая стоимость на больших объемах | Тестирование облачных и мобильных приложений |
| Artillery | Простота настройки, высокоскоростное тестирование, открытый исходный код | Меньше функций по сравнению с крупными решениями, сложность в настройке отчетности | Веб-приложения, микросервисы, серверы |
| NeoLoad | Интуитивно понятный интерфейс, интеграция с CI/CD, подробные отчеты | Высокая стоимость лицензии, ограниченная поддержка некоторых протоколов | Корпоративные приложения, веб-приложения, облачные сервисы |

Одним из методов оценки производительности микросервисной архитектуры является тестирование пропускной способности с использованием инструментов, таких как Apache JMeter или Gatling. В ходе экспериментов увеличивается количество запросов к сервису, что позволяет наблюдать изменение времени отклика и обработанных запросов при различных уровнях нагрузки. Важно учитывать пиковые нагрузки, характерные для транзакционной активности, и постоянные нагрузки, связанные с повседневными операциями. Результаты тестов показывают, что увеличение числа сервисов не всегда приводит к линейному улучшению производительности, что требует оптимизации взаимодействия компонентов.

Кроме того, анализ времени отклика и латентности в распределенных вычислениях проводится через тесты с измерением задержек между компонентами

системы. Использование сервисных мешей, таких как Istio, и API Gateway помогает снизить сетевые задержки, улучшая общее время отклика. Для оценки эффективности потребления ресурсов проводятся тесты мониторинга CPU и памяти, выявляя избыточное использование и оптимизируя распределение ресурсов. Также анализируются стратегии масштабирования, такие как горизонтальное и вертикальное, а также автоматическое масштабирование.

## Заключение

Оптимизация производительности микросервисной архитектуры в финтех-проектах является ключевым аспектом успешного функционирования современных финансовых систем. Переход от монолитных решений к микросервисам способствует улучшению гибкости, масштабируемости и скорости внедрения новых сервисов, однако приводит к новым вызовам в области производительности. Важно учитывать, что микросервисная архитектура требует применения комплексных методов оценки производительности, включая анализ пропускной способности, времени отклика и использования ресурсов, с учетом специфики финансовых приложений.

Рассмотренные архитектурные решения, такие как REST и gRPC, использование асинхронных очередей сообщений и сервис-мешей, напрямую влияют на производительность и надежность системы. Для оптимизации производительности необходимо не только внедрять эффективные механизмы межсервисного взаимодействия, но и обеспечить балансировку нагрузки, динамическое масштабирование и надежное управление состоянием.

Исследования показывают, что правильный выбор архитектурных решений и инструментов для мониторинга и тестирования позволяет значительно повысить эффективность работы микросервисных платформ в финтехе, обеспечивая их стабильность, минимальные задержки и высокую пропускную способность, что важно для успешного функционирования финансовых сервисов.

## СПИСОК ИСТОЧНИКОВ

1. Финансовые технологии как инструмент управления долговой нагрузкой населения / А.Д. Захаров, В.Н. Грепан, И.Ю. Благова [и др.] // Первый экономический журнал. – 2024. – № 9 (351). – С. 87–96.

2. Sun Y. A Comparative Study of Application Performance and Resource Consumption between Monolithic and Microservice Architectures [Electronic resource] // Helsinki University Library. – URL: https://helda.helsinki.fi/items/c9117b35-318d-4e42-8c60-38f607e0d3d1 [Accessed 23rd December 2024].

3. Lytvynov O.A. Critical causal events in systems based on CQRS with event sourcing architecture / O.A. Lytvynov, D.L. Hruzin // Radio Electronics, Computer Science, Control. – 2024. – No. 3. – P. 119–143.

4. Sandberg F. Evaluating OpenTelemetry's Impact on Performance in Microservice Architectures [Electronic resource]. – URL: https://umu.diva-portal.org/smash/get/diva2:1877027/FULLTEXT01.pdf [Accessed 23rd December 2024].

5. Кузнецов И.А. Оптимизация распределенных систем для мобильных приложений: улучшение производительности и масштабируемости / И.А. Кузнецов // Инновационная наука. – 2024. – № 5–1. – С. 52–57.

6. Leg R. gRPC vs REST speed comparison [Electronic resource] // SHIFT ASIA | Dev Blog. – URL: https://blog.shiftasia.com/grpc-vs-rest-speed-comparation/ [Accessed 23rd December 2024].

7. Werner S. A reference architecture for serverless big data processing / S. Werner, S. Tai // Future Generation Computer Systems. – 2024. – Vol. 155. – P. 179–192.

8. Behera T.K. From Reactive to Proactive: Predicting and Optimizing Performance for Competitive Advantage / T.K. Behera, D.M. Dave // Transforming Industry using Digital Twin Technology. – Cham: Springer, 2024. – P. 69–93.

9. Performance and Scalability [Electronic resource] // Istio. – URL: https://istio.io/latest/docs/ops/deployment/performance-and-scalability/ [Accessed 23rd December 2024].

## ИНФОРМАЦИЯ ОБ АВТОРЕ

**Болгов Сергей Николаевич,** специалист, Мурманский арктический университет, Мурманск, Россия.
*e-mail:* bolgov_sergei@rambler.ru