Программные системы и вычислительные методы
Правильная ссылка на статью:
Чикалева Ю.С. Анализ гранулярности микросервисов: эффективность архитектурных подходов // Программные системы и вычислительные методы. 2025. № 2. DOI: 10.7256/2454-0714.2025.2.74386 EDN: NCFQFE URL: https://nbpublish.com/library_read_article.php?id=74386

Анализ гранулярности микросервисов: эффективность архитектурных подходов

Чикалева Юлия Сергеевна

ORCID: 0009-0009-7548-594X

студент; Мегафакультет компьютерных технологий и управления; Национальный исследовательский университет ИТМО

197101, Россия, г. Санкт-Петербург, Петроградский р-н, Кронверкский пр-кт, д. 49

y.chikaleva99@mail.ru

Статья из рубрики "Математическое моделирование и вычислительный эксперимент"

DOI:

10.7256/2454-0714.2025.2.74386

EDN:

NCFQFE

Дата направления статьи в редакцию:

10-05-2025

Дата публикации:

29-05-2025

Аннотация: Современные информационные системы требуют масштабируемых архитектур для обработки больших данных и обеспечения доступности. Микросервисная архитектура, разделяющая приложения на автономные сервисы по бизнес-функциям, решает эти задачи. Однако оптимальная гранулярность микросервисов влияет на производительность, масштабируемость и управляемость. Неоптимальная декомпозиция приводит к антипаттернам, таким как избыточная мелкость или косметическая микросервисность, усложняя сопровождение. Цель исследования — сравнительный анализ методов определения гранулярности микросервисов для выявления подходов, обеспечивающих баланс производительности, гибкости и управляемости в высоконагруженных системах. Объект исследования – микросервисная архитектура высоконагруженных информационных систем, включая их структурные и

функциональные характеристики, определяемые методами декомпозиции на автономные сервисы. Предмет исследования – методы определения гранулярности микросервисов (монолитная архитектура, Domain-Driven Design, Data-Driven Approach, Monolith to Microservices Approach). Применён экспериментальный подход, включающий реализацию приложения Task Manager в четырёх архитектурных конфигурациях. Нагрузочное тестирование проводилось с использованием Apache JMeter при нагрузке 1000 пользователей. Метрики производительности (время отклика, пропускная способность, CPU), доступности, масштабируемости, безопасности и согласованности собраны через Prometheus и обработаны с вычислением средних значений и стандартного отклонения. Научная новизна исследования заключается в разработке унифицированной методологии количественного анализа методов гранулярности микросервисов (монолит, DDD, Data-Driven, Monolith to Microservices), основанной на метриках (время отклика, пропускная способность, CPU, доступность, запуск, безопасность, ошибки), адаптированных для высоконагруженных систем. В отличие от качественных исследований, работа предлагает комплексный подход, включая реализацию приложения Task Manager и нагрузочное тестирование (Apache JMeter, Prometheus), решая проблему выбора оптимальной декомпозиции для повышения производительности и масштабируемости. Методология применима для автоматизации оценки архитектур в CI/CD. Монолит показал минимальное время отклика (0.76 с) и пропускную способность (282.5 запросов/с), но ограничен масштабируемостью. Data-Driven обеспечивает согласованность, DDD эффективен для сложных доменов, Monolith to Microservices имеет низкую производительность (15.99 с) из-за перегрузки авторизации. Ограничение — хост-система (8 ГБ RAM), снижающая масштабируемость. Рекомендации: оптимизировать сетевые вызовы в DDD, доступ к данным в Data-Driven, декомпозицию в Monolith to Microservices.

Ключевые слова:

микросервисная архитектура, гранулярность микросервисов, Domain-Driven Design, Data-Driven Approach, Monolith to Microservices, производительность, масштабируемость, согласованность данных, экспериментальный анализ, высоконагруженные системы

Введение

Современные информационные системы сталкиваются с необходимостью обработки больших объемов данных, обеспечения высокой доступности и быстрого вывода продуктов на рынок. Микросервисная архитектура, разделяющая приложение на автономные сервисы, ориентированные на бизнес-функции, предоставляет разработчикам инструменты для создания масштабируемых и гибких систем [1]. Однако неоптимальный выбор гранулярности микросервисов может привести к избыточной сложности, увеличению коммуникационных издержек и снижению производительности [2].

Актуальность исследования обусловлена широким распространением микросервисной архитектуры и связанными с ней вызовами. Неоптимальная декомпозиция системы может привести к антипаттернам, таким как избыточная мелкость микросервисов или косметическая микросервисность, что усложняет масштабирование и сопровождение [3]. Наблюдается недостаток комплексных сравнений методов определения гранулярности микросервисов, что усложняет выбор оптимального подхода. В то же время оптимальная

гранулярность позволяет достичь баланса между гибкостью, производительностью и управляемостью.

Цель исследования заключается в анализе и сравнении методов определения гранулярности микросервисов с целью выявления наиболее эффективных подходов для проектирования высоконагруженных систем.

Этапы исследования

Для проведения сравнительного анализа методов определения гранулярности микросервисов были выделены следующие этапы:

1. **Выбор тестового приложения**: в качестве тестового приложения был разработан Task Manager, предоставляющий функциональность управления задачами и пользователями. Приложение включает регистрацию, аутентификацию, создание и редактирование задач, что делает его подходящим для моделирования реальных высоконагруженных сценариев.

2. **Разработка тестового приложения**: приложение было реализовано с использованием Java Spring Boot и MySQL. Рассматривались четыре архитектурные конфигурации: монолитная архитектура (для сравнения), Domain-Driven Design (DDD), Data-Driven Approach и Monolith to Microservices Approach. Каждая конфигурация представляла различный подход к декомпозиции системы.

3. **Настройка тестового окружения**: для обеспечения объективности тестирования использовалась контейнеризация на основе Docker. Метрики собирались с помощью системы мониторинга Prometheus, а нагрузочное тестирование проводилось с использованием Apache JMeter. Тестовое окружение обеспечивало изоляцию и минимизацию внешних факторов.

4. **Проведение тестов**: тестирование включало измерение ключевых метрик (время отклика, пропускная способность, использование CPU, доступность, время запуска экземпляров, уровень безопасности, процент ошибок) для каждой архитектурной конфигурации. Тесты проводились в условиях последовательной отправки запросов для оценки производительности и устойчивости.

5. **Анализ результатов**: собранные данные были проанализированы для выявления преимуществ и недостатков каждого метода декомпозиции. На основе анализа сформулированы рекомендации по применению методов в зависимости от проектных требований.

Тестовое приложение

Результаты исследования зависят от объективности выбора тестового приложения. Task Manager был спроектирован как приложение, моделирующее реальные сценарии высоконагруженных систем. Основной функционал включает:

· Регистрацию и аутентификацию пользователей.

· Создание, редактирование и удаление задач.

· Управление статусом задач и их назначением.

Приложение реализовано на Java Spring Boot с использованием RESTful API. Для хранения данных применялась реляционная база данных MySQL, обеспечивающая

надежность и производительность. Архитектурно приложение поддерживает различные конфигурации:

· **Монолитная архитектура**: вся функциональность объединена в одном сервисе с общей базой данных (рис. 1). Монолитная архитектура не предполагает декомпозиции, поэтому вся бизнес-логика (управление пользователями, задачами и их связями) реализована в едином приложении [4]. Это позволяет минимизировать сетевые накладные расходы и упростить управление данными, что делает монолит эталоном для сравнения с микросервисными подходами [5]. Общая база данных MySQL обеспечивает централизованное хранение всех данных, что соответствует традиционной монолитной модели [6].

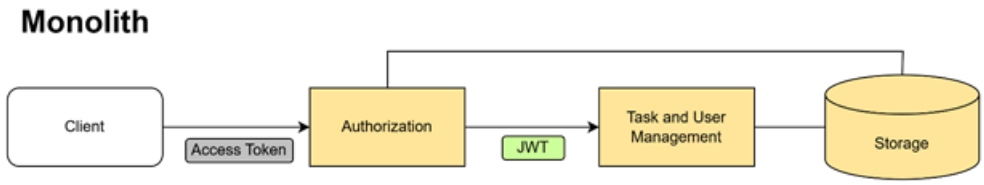

Рисунок 1. Схема монолитной архитектуры приложения Task Manager: единый сервис с общей базой данных MySQL, обеспечивающий управление пользователями и задачами.

Domain-Driven Design (DDD): Система разделена на три микросервиса — User Identity Service, Task Management Service, Task Assignment Service — на основе доменных областей (рис. 2). Метод DDD фокусируется на выделении границ домена (bounded context) в соответствии с бизнес-логикой [7]. User Identity Service отвечает за аутентификацию и управление данными пользователей, что представляет домен идентификации. Task Management Service управляет жизненным циклом задач (создание, редактирование, удаление), охватывая домен управления задачами [8]. Task Assignment Service связывает задачи и пользователей, представляя домен назначения задач. Такое разделение обеспечивает высокую модульность [9] и изоляцию бизнес-логики, что упрощает поддержку и масштабирование каждого домена независимо [10].

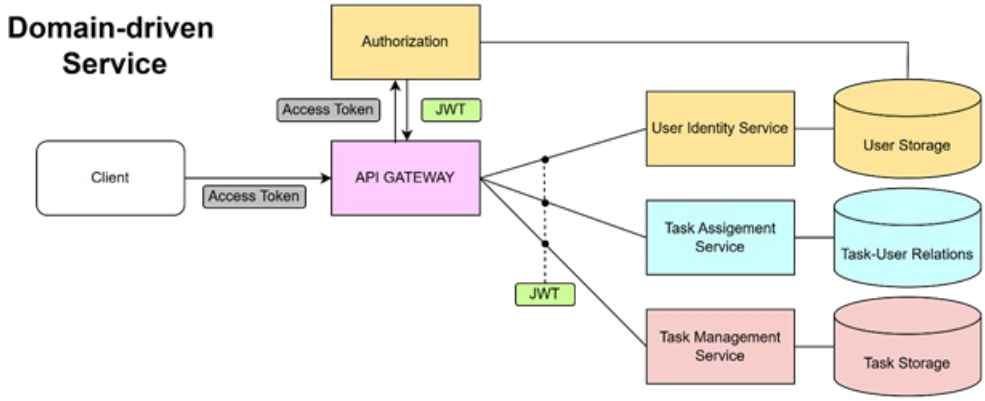

Рисунок 2. Схема архитектуры Task Manager на основе DDD: три микросервиса (User Identity Service, Task Management Service, Task Assignment Service), разделенные по доменным областям, взаимодействующие через REST API.

Data-Driven Approach: два микросервиса (Task Service, User Service) с изолированными базами данных (рис. 3). Подход, основанный на данных, предполагает декомпозицию системы на основе структуры данных и их независимости [11]. Task Service управляет данными задач, включая их создание, редактирование и статус, и использует

собственную базу данных MySQL для хранения информации о задачах. User Service отвечает за данные пользователей и аутентификацию, также с отдельной базой данных MySQL. Разделение по данным минимизирует зависимости между сервисами, обеспечивает автономность хранения и обработки данных, а также позволяет выбирать оптимальные технологии для каждого хранилища. Это снижает риск компрометации всей системы при атаке на одно хранилище, но требует синхронизации данных между сервисами.

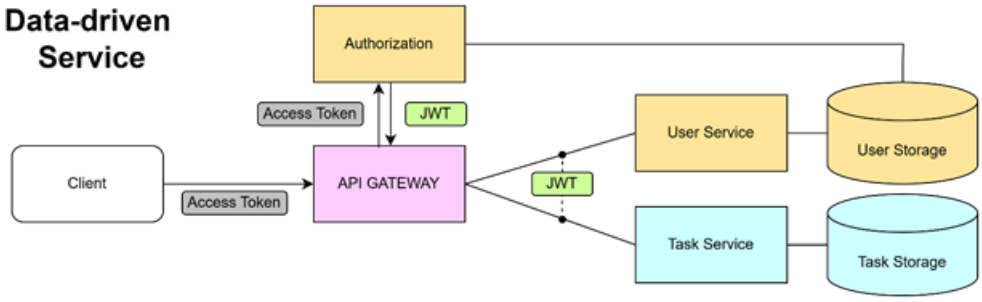

Рисунок 3. Схема архитектуры Task Manager на основе Data-Driven Approach: два микросервиса (Task Service, User Service) с изолированными базами данных MySQL, обеспечивающие независимость данных.

Monolith to Microservices Approach: выделен Authorization Service, остальная функциональность осталась в монолите (рис. 4). Метод перехода от монолита к микросервисам предполагает постепенную декомпозицию, начиная с выделения наиболее критичных и часто используемых функций [12]. Authorization Service был выбран для выделения, так как аутентификация пользователей является ключевой функцией, требующей высокой производительности и безопасности. Этот сервис управляет аутентификацией и работает с собственной базой данных MySQL, что позволяет изолировать критичную функциональность и снизить риски при миграции. Остальная функциональность (управление задачами и их назначением) осталась в монолите, что минимизирует изменения на начальном этапе перехода, но ограничивает автономность системы до полной декомпозиции [13].

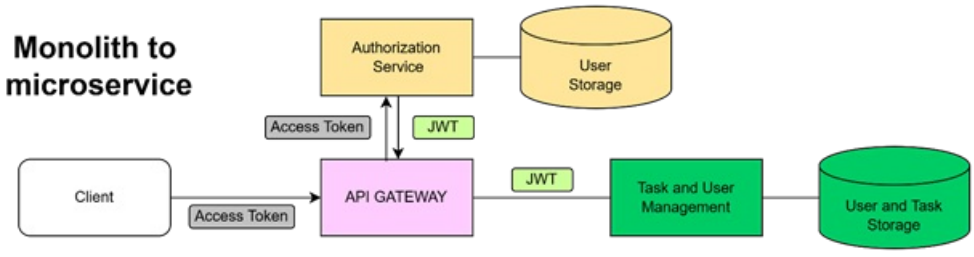

Рисунок 4. Схема архитектуры Task Manager на основе Monolith to Microservices Approach: выделен Authorization Service с собственной базой данных MySQL, остальная функциональность реализована в монолите.

Исходный код приложения доступен в репозитории на GitHub: [https://github.com/Yuliya3003/Microservices_research.git].

Методика тестирования

Тестирование проводилось в контролируемом окружении для обеспечения воспроизводимости и объективности результатов. Все архитектурные конфигурации приложения Task Manager (монолитная архитектура, Domain-Driven Design, Data-Driven

Approach, Monolith to Microservices Approach) были развернуты с использованием контейнеризации на основе Docker, что обеспечивало изоляцию сервисов и минимизацию влияния внешних факторов. Для сбора метрик использовалась система мониторинга Prometheus, а нагрузочное тестирование проводилось с помощью инструмента Apache JMeter. Тесты были направлены на оценку производительности, доступности, масштабируемости, безопасности и согласованности каждой конфигурации [14].

Процесс тестирования

Тестирование включало последовательную отправку HTTP-запросов к RESTful API приложения Task Manager для моделирования высоконагруженных сценариев. Процесс тестирования состоял из следующих этапов:

1. **Подготовка тестового окружения**:

o Все архитектурные конфигурации приложения Task Manager были развернуты в контейнерах Docker на сервере с операционной системой Microsoft Windows 11 Pro (версия 10.0.22631), 8 ГБ оперативной памяти и 4-ядерным процессором AMD Ryzen 3 5300U.

o Каждый контейнер содержал отдельный экземпляр сервиса (или монолитного приложения) и связанную базу данных MySQL. Для микросервисных конфигураций использовались отдельные базы данных для каждого сервиса, если это предусматривалось методом декомпозиции.

o Prometheus был настроен для сбора метрик с экспортеров, встроенных в приложение (Spring Boot Actuator).

2. **Конфигурация нагрузочного тестирования**:

o Нагрузочное тестирование проводилось с использованием Apache JMeter версии 5.6.3. Тестовый план моделировал высокую нагрузку на приложение путем отправки HTTP-запросов.

o Параметры теста:

§ **Количество потоков**: 1000 виртуальных пользователей, имитирующих одновременные запросы.

§ **Время разгона (Ramp-up)**: 10 секунд, чтобы постепенно увеличить нагрузку и избежать резких пиков.

§ **Количество итераций**: 10 циклов выполнения запросов для каждого потока, что обеспечивало достаточный объем данных для анализа.

o Тестовый план включал сбор результатов через компоненты Summary Report и View Results Tree, которые фиксировали время отклика, пропускную способность, статусы ответов и ошибки.

3. **Сбор и обработка метрик**:

o Метрики собирались в реальном времени с использованием Prometheus, настроенного на интервал сбора данных 1 секунда.

o После завершения тестов данные экспортировались из Prometheus для дальнейшего

анализа. Для расчета средних значений и построения графиков использовались запросы PromQL (Prometheus Query Language).

Описание метрик и их измерение

Для выбора характеристик и метрик, используемых в сравнительном анализе методов определения гранулярности микросервисов, был проведён систематический обзор 56 научных статей, опубликованных в рецензируемых изданиях и посвящённых измерению эффективности программных систем, включая микросервисные архитектуры [15]. В рамках анализа были идентифицированы наиболее часто встречающиеся характеристики качества [16], такие как производительность, доступность, масштабируемость, безопасность и согласованность, которые имеют критическое значение для высоконагруженных распределённых систем [17]. Частота упоминания данных характеристик в рассмотренных источниках представлена на рисунке 5.

Рисунок 5. Частота встречаемости исследуемых критериев в научных статьях.

Выбор метрик осуществлялся с учётом их измеримости с использованием системы мониторинга Prometheus, что обеспечивало высокую точность и воспроизводимость результатов [18]. Перечень метрик, их определение и методы измерения представлены в Таблице 1.

Таблица 1. Количественные метрики для оценки характеристик качества системы

Характеристика	Метрика	Измерение
Производительность	Время отклика, с	$\frac{http\_server\_requests\_seconds\_sum}{http\_server\_requests\_seconds\_count}$
	CPU	process_cpu_usage
	Пропускная способность, rec/s	rate(http_server_requests_seconds_count[5m])
Доступность	Среднее время	avg_over_time(up{job="service_name"}[5m])

Доступность	безотказной работы, %	* 100%
Масштабируемость	Время запуска экземпляров, с	application_started_time_seconds
Безопасность	Уровень безопасности	калькулятор CVSS 4.0
Согласованность	Процент ошибок, %	$\frac{rate(http\_server\_requests\_seconds\_count\{status=\sim"4..|5.."\}[5m])}{rate(http\_server\_requests\_seconds\_count[5m]) * 100}$

Результаты тестирования

Результаты измерений представлены в Таблице 2.

Монолитная архитектура продемонстрировала высокую производительность, обусловленную отсутствием сетевых задержек, характерных для распределённых систем [19]. Среднее время отклика составило 0.76 секунды, а пропускная способность достигла 282.5 запросов в секунду при минимальном использовании CPU (0.0055). Однако масштабируемость конфигурации ограничена, о чём свидетельствует относительно высокое время запуска экземпляров (14.86 секунды), что указывает на потенциальные сложности при горизонтальном масштабировании.

Domain-Driven Design (DDD) обеспечивает модульность за счёт разделения системы на независимые доменные области, что упрощает сопровождение сложных бизнес-логики. Тем не менее, межсервисные вызовы привели к увеличению времени отклика до 1.07 секунды и снижению пропускной способности до 27.6 запросов в секунду. Использование CPU (0.1698) также оказалось выше, чем у монолита, что отражает дополнительные накладные расходы на координацию сервисов.

Data-Driven Approach выделяется высокой согласованностью данных благодаря изоляции баз данных для каждого сервиса, что минимизирует конфликты при параллельной обработке. Однако изоляция баз привела к значительному увеличению времени отклика (5.49 секунды), что указывает на необходимость оптимизации запросов к данным. Пропускная способность составила 35.6 запросов в секунду, а использование CPU (0.1704) оказалось сопоставимым с DDD. Время запуска экземпляров (24.37 секунды) отражает умеренные затраты на инициализацию изолированных компонентов, что согласуется с выводами о влиянии гранулярности на масштабируемость микросервисных систем [20].

Monolith to Microservices Approach показал наименее удовлетворительные результаты, что обусловлено неполной декомпозицией системы. Выделение Authorization Service привело к высокой нагрузке на этот компонент, что проявилось в максимальном времени отклика (15.99 секунды) и низкой пропускной способности (33.34 запроса в секунду). Использование CPU (0.1778) было наивысшим среди всех конфигураций, а время запуска экземпляров (33.68 секунды) указывает на значительные затраты на инициализацию частично декомпозированной системы.

Таблица 2. Сравнение результатов тестирования

		Монолит	Domain-	Data-	Monolith to

		Монолит	Driven	Driven	microservice
Производительность	Время отклика, с	0,76	1,07	5,49	15,99
	CPU	0,0055	0,1698	0,1704	0,1778
	Пропускная способность, rec/s	282,5	27,6	35,6	33,34
Доступность	Среднее время безотказной работы, %	100	100	100	100
Масштабируемость	Время запуска экземпляров, с	14,86	31,86	24,37	33,68
Безопасность	Уровень безопасности	5,4	5,4	5,4	5,4
Согласованность	Процент ошибок, %	0	0	0	0

Монолитная архитектура демонстрирует наилучшую производительность, но ограничена в гибкости. Data-Driven Approach выделяется высокой согласованностью данных, DDD подходит для сложных доменов, а Monolith to Microservices требует дальнейшей декомпозиции.

Анализ результатов

Проведённое исследование подтвердило существенное влияние метода декомпозиции на характеристики качества системы, измеренные в ходе нагрузочного тестирования с использованием Apache JMeter и мониторинга через Prometheus. Анализ данных, представленных в Таблице 2, выявил преимущества и ограничения каждой архитектурной конфигурации приложения Task Manager.

Монолитная архитектура продемонстрировала наивысшую производительность, выраженную в минимальном времени отклика (0.76 секунды) и высокой пропускной способности (282.5 запросов в секунду), что обусловлено отсутствием сетевых накладных расходов. Однако ограниченная масштабируемость, проявляющаяся в относительно высоком времени запуска экземпляров (14.86 секунды), делает её менее подходящей для систем, требующих быстрого горизонтального расширения.

Архитектура на основе Domain-Driven Design (DDD) обеспечивает модульность, упрощающую сопровождение сложных доменных моделей. Тем не менее, межсервисные взаимодействия увеличили время отклика до 1.07 секунды и снизили пропускную способность до 27.6 запросов в секунду, что указывает на необходимость оптимизации сетевых вызовов.

Data-Driven Approach показал высокую согласованность данных за счёт изоляции баз данных, минимизирующей конфликты при параллельных операциях. Однако значительное время отклика (5.49 секунды) подчёркивает потребность в оптимизации

доступа к данным, несмотря на умеренную пропускную способность (35.6 запросов в секунду) и приемлемое время запуска экземпляров (24.37 секунды).

Monolith to Microservices Approach оказался наименее эффективным из-за неполной декомпозиции, что привело к перегрузке Authorization Service. Максимальное время отклика (15.99 секунды) и низкая пропускная способность (33.34 запроса в секунду) отражают ограничения конфигурации, а высокое время запуска экземпляров (33.68 секунды) подчёркивает сложности масштабирования.

На основе анализа результатов предлагаются следующие рекомендации:

1. Использовать монолитную архитектуру для систем с приоритетами производительности и ограниченными требованиями к масштабируемости.

2 . Применять DDD для проектов со сложной доменной логикой, уделяя внимание оптимизации межсервисных взаимодействий.

3 . Выбирать Data-Driven Approach для систем, требующих высокой согласованности данных, с акцентом на оптимизацию запросов к базам данных.

4 . Проводить тщательное планирование декомпозиции при переходе от монолита к микросервисам, чтобы избежать перегрузки отдельных компонентов.

Заключение

В ходе исследования были систематизированы методы определения гранулярности микросервисов, разработано приложение Task Manager и проведено тестирование четырех архитектурных подходов. Полученные данные подтвердили влияние гранулярности на производительность, масштабируемость и согласованность системы.

Результаты исследования могут служить основой для дальнейших работ в области оптимизации микросервисных архитектур. Перспективные направления включают:

- Исследование асинхронных паттернов (Saga, Event Sourcing) для снижения сетевых накладных расходов.
- Анализ влияния оркестрации (Kubernetes) на масштабируемость микросервисов.
- Разработка автоматизированных инструментов для оценки гранулярности в CI/CD-процессах.

Библиография

1. Ворсин В. А. Микросервисная архитектура бизнес-приложений: перспективы и проблемы // GLOBUS. 2020. Вып. 4 (50). С. 51-53. EDN: TYXBZZ.
2. Raj P., Vanga S., Chaudhary A. Microservices Security // CloudINative Computing. 2022. DOI: 10.1002/9781119814795.ch14.
3. Зиборев А. В. Антипаттерны построения микросервисных приложений в высоконагруженных проектах // UNIVERSUM: Технические науки. 2023. Вып. 11 (116). Ноя. 2023. doi: 10.32743/UniTech.2023.116.11.16204. EDN: HBLBIO.
4. Newman S. Building Microservices. 2nd ed. Sebastopol, CA: O'Reilly Media, 2025.
5. Blinowski G. и др. Monolithic vs. Microservice Architecture: Performance and Scalability Evaluation // IEEE Access. 2022. DOI: 10.1109/ACCESS.2022.3152803. EDN: JHKSCZ.
6. Richardson C. Microservice Architecture Pattern. 2020. Available at: https://microservices.io/patterns/microservices.html (accessed: Jan. 05, 2025).
7. Momil S., Qaisar A. Transition Strategies from Monolithic to Microservices Architectures: A Domain-Driven Approach and Case Study // VAWKUM Transactions on Computer Sciences.

2024. Vol. 12, No. 1. Pp. 94-110. DOI: 10.21015/vtcs.v12i1.1808. EDN: ABOYUL.
8. Радостев Д. К., Никитина Е. Ю. Стратегия миграции программного кода из монолитной архитектуры в микросервисы // Вестник Пермского Университета. 2021. Вып. 2 (53). С. 65-68. doi: 10.17072/1993-0550-2021-2-65-68. EDN: DSNKOZ.
9. Jordanov J., Petrov P. Domain Driven Design Approaches in Cloud Native Service Architecture // TEM Journal. 2023. Vol. 12, Issue 4. C. 1985-1994. ISSN 2217-8309. DOI: 10.18421/TEM124-09. EDN: MMRBFA.
10. Vular H., Koyuncu M. Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice? // IEEE Access. 2021. Vol. 9. Pp. 27960-27971. DOI: 10.1109/ACCESS.2021.306089.
11. Hasselbring W., Steinacker G. Microservice Architectures for Scalability, Agility and Reliability in E-Commerce // 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). Gothenburg: IEEE, 2017. P. 46-49. DOI: 10.1109/ICSAW.2017.11.
12. Schmidt R. A., Thiry M. Microservices Identification Strategies: A Review Focused on Model-Driven Engineering and Domain-Driven Design Approaches // 2020 15th Iberian Conference on Information Systems and Technologies (CISTI). Seville: IEEE, 2020. P. 1-6. DOI: 10.23919/CISTI49556.2020.9141150.
13. Christudas B. Practical Microservices Architectural Patterns. New York: Apress, 2019. DOI: 10.1007/978-1-4842-4501-9.
14. Sriraman A., Wenisch T. F. μSuite: A Benchmark Suite for Microservices // 2018 IEEE International Symposium on Workload Characterization (IISWC). Raleigh: IEEE, 2018. P. 1-12. DOI: 10.1109/IISWC.2018.8573515.
15. Henning S., Hasselbring W. Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud // Journal of Systems and Software. 2024. Vol. 208. [Online]. Available: https://doi.org/10.1016/j.jss.2023.111879. EDN: RVNVGK.
16. Bjørndal N., & Pires de Araújo, etc. Benchmarks and performance metrics for assessing the migration to microservice-based architectures // The Journal of Object Technology. August 2021. DOI: 10.5381/jot.
17. Артамонов И. В. Показатели производительности микросервисных систем // Вестник НГИЭИ. 2018. Вып. 8 (87). С. 24-33. EDN: XYTGKT.
18. Waseem M., Liang P., Shahin M., et al. Design, Monitoring, and Testing of Microservices Systems: The Practitioners' Perspective // Journal of Systems and Software. 2021. Vol. 182. Article 111061. DOI: 10.1016/j.jss.2021.111061. EDN: ATPBYU.
19. Ramu V. Performance Impact of Microservices Architecture // Rev. Contemp. Sci. Acad. Stud. 2023. Vol. 3. P. 1-6.
20. Hassan S., Bahsoon R., & Buyya R. Systematic scalability analysis for microservices granularity adaptation design decisions // Software: Practice and Experience. 2022. Vol. 52, No. 6. DOI: 10.1002/spe.3069. EDN: MOLQOU.

Результаты процедуры рецензирования статьи

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

Представленная статья на тему «Анализ гранулярности микросервисов: эффективность архитектурных подходов» соответствует тематике журнала «Программные системы и вычислительные методы» и посвящена актуальному вопросу. Актуальность исследования обусловлена широким распространением микросервисной архитектуры и связанными с ней вызовами. Неоптимальная декомпозиция системы может привести к антипаттернам, таким как избыточная мелкость микросервисов или косметическая микросервисность,

что усложняет масштабирование и сопровождение. Наблюдается недостаток комплексных сравнений методов определения гранулярности микросервисов, что усложняет выбор оптимального подхода. В то же время оптимальная гранулярность позволяет достичь баланса между гибкостью, производительностью и управляемостью.

В качестве цели авторы указывают анализ и сравнение методов определения гранулярности микросервисов с целью выявления наиболее эффективных подходов для проектирования высоконагруженных систем.

Для проведения сравнительного анализа методов определения гранулярности микросервисов авторами были выделены следующие этапы:

1. Выбор тестового приложения.
2. Разработка тестового приложения.
3. Настройка тестового окружения.
4. Проведение тестов.
5. Анализ результатов.

Для выбора характеристик и метрик, используемых в сравнительном анализе методов определения гранулярности микросервисов, авторами был проведён систематический обзор 56 научных статей, опубликованных в рецензируемых изданиях и посвящённых измерению эффективности программных систем, включая микросервисные архитектуры. В рамках анализа были идентифицированы наиболее часто встречающиеся характеристики качества, такие как производительность, доступность, масштабируемость, безопасность и согласованность, которые имеют критическое значение для высоконагруженных распределённых систем.

Также, проведённое авторами исследование подтвердило существенное влияние метода декомпозиции на характеристики качества системы, измеренные в ходе нагрузочного тестирования с использованием Apache JMeter и мониторинга через Prometheus.

Результаты исследования представлены в табличном и графическом виде.

В списке литературы представлены зарубежные источники по теме исследования. Стиль и язык изложения материала является достаточно доступным для широкого круга читателей. Статья по объему соответствует рекомендуемому объему от 12 000 знаков.

Практическая значимость статьи четко обоснована. Результаты исследования могут служить основой для дальнейших работ в области оптимизации микросервисных архитектур. По мнению авторов перспективными направлениями являются:

- Исследование асинхронных паттернов для снижения сетевых накладных расходов.
- Анализ влияния оркестрации на масштабируемость микросервисов.
- Разработка автоматизированных инструментов для оценки гранулярности в CI/CD-процессах.

Статья достаточно структурирована - в наличии введение, заключение, внутреннее членение основной части.

К недостаткам можно отнести следующие моменты: из содержания статьи не прослеживается научная новизна. Отсутствует четкое выделение предмета, объекта исследования.

Рекомендуется четко обозначить научную новизну исследования, сформулировать предмет, объект. Также будет целесообразным расширить список литературы, указав работы отечественных авторов.

Статья «Анализ гранулярности микросервисов: эффективность архитектурных подходов» требует доработки по указанным выше замечаниям. После внесения поправок рекомендуется к повторному рассмотрению редакцией рецензируемого научного журнала.

Результаты процедуры повторного рецензирования статьи

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

Рецензируемая статья посвящена одному из ключевых вопросов проектирования распределённых систем – анализу гранулярности микросервисов.
Методология исследования базируется на разработке тестового программного приложения и проведении тестирование четырех архитектурных подходов: монолитной архитектуры, архитектуры на основе Domain-Driven Design, Data-Driven Approach и Monolith to Microservices Approach.
Актуальность работы авторы связывают с широким распространением микросервисной архитектуры, с тем, что неоптимальный выбор гранулярности микросервисов может привести к избыточной сложности, увеличению коммуникационных издержек и снижению производительности.
Научная новизна работы состоит в систематизации методов определения гранулярности микросервисов, полученных результатах сравнения различных четырех таких методов, в подтверждении влияния гранулярности на производительность, масштабируемость и согласованность системы и в выявлении эффективных подходов для проектирования высоконагруженных систем.
В тексте публикации структурно выделены следующие разделы: Введение, Этапы исследования, Тестовое приложение, Методика тестирования, Процесс тестирования, Описание метрик и их измерение, Результаты тестирования, Анализ результатов, Заключение и Библиография.
В публикации освещены этапы проведения сравнительного анализа методов определения гранулярности микросервисов: выбор тестового приложения, разработка тестового приложения, настройка тестового окружения, проведение тестов, анализ результатов. Авторами проведено изучение публикаций, посвященных измерению эффективности программных систем, отражена частота встречаемости исследуемых критериев в научных статьях, выбраны метрики для измерения таких характеристик как производительность, доступность, масштабируемость, безопасность и согласованность. Тестирование включало измерение ключевых метрик (время отклика, пропускная способность, использование центрального процессора, доступность, время запуска экземпляров, уровень безопасности, процент ошибок) для каждой архитектурной конфигурации. Отмечено, что монолитная архитектура демонстрирует наилучшую производительность, но ограничена в гибкости. Data-Driven Approach выделяется высокой согласованностью данных, DDD подходит для сложных доменов, а Monolith to Microservices требует дальнейшей декомпозиции. Статья иллюстрирован 5 рисунками, содержит 2 таблицы.
Библиографический список включает 13 источников: работы российских и зарубежных авторов по рассматриваемой тематике на русском и иностранном языках. В тексте публикации имеются адресные отсылки к списку литературы, подтверждающие наличие апелляции к оппонентам.
Из замечаний стоит отметить, что названия таблиц почему-то приведены не перед ними, как это предусмотрено правилами оформления и ГОСТами, а после таблиц; авторами не соблюдены принятые редакцией Правила оформления списка литературы по количеству и категориям источников: «Рекомендованный объем списка литературы … – не менее 20 источников, который должен содержать: … не менее половины работ, изданных в последние 3 года».
Тема статьи актуальна, отражает результаты проведенного авторами исследования, соответствует тематике журнала «Программные системы и вычислительные методы»,

может вызвать интерес у определенного круга IT-специалистов, но материал нуждается в доработке.

Результаты процедуры окончательного рецензирования статьи

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

Статья посвящена важной и актуальной проблеме определения оптимальной гранулярности микросервисов в современных информационных системах. Автор проводит детальное исследование, сравнивая различные архитектурные подходы, включая Domain-Driven Design, Data-Driven Approach и метод постепенного перехода от монолита к микросервисам. В качестве инструмента анализа используется разработанное тестовое приложение Task Manager, что позволяет наглядно оценить влияние каждого подхода на ключевые характеристики системы, такие как производительность, масштабируемость и согласованность данных.
Методологическая основа исследования заслуживает высокой оценки. Автор применяет комплексный подход, сочетающий теоретический анализ с практической реализацией. Тестовое приложение Task Manager разработано с поддержкой четырех архитектурных конфигураций, что обеспечивает объективность сравнения. Нагрузочное тестирование проводится с использованием Apache JMeter, а сбор метрик осуществляется через систему мониторинга Prometheus. Такой подход позволяет получить достоверные данные о времени отклика, пропускной способности, использовании CPU и других критически важных показателях. Результаты представлены в виде наглядных таблиц и графиков, что значительно облегчает восприятие и анализ информации.
Актуальность исследования не вызывает сомнений. В условиях стремительного роста популярности микросервисных архитектур вопросы оптимизации гранулярности, снижения коммуникационных издержек и обеспечения масштабируемости становятся все более значимыми. Статья заполняет существенный пробел в области сравнительного анализа методов декомпозиции, предлагая практические решения для разработчиков и архитекторов. Это делает работу особенно ценной для специалистов, занимающихся проектированием высоконагруженных распределенных систем.
Научная новизна исследования проявляется в систематизации методов определения гранулярности микросервисов и их эмпирической проверке на реальном тестовом приложении. Автор не только сравнивает различные подходы, но и формулирует четкие рекомендации по их применению в зависимости от конкретных требований проекта. Например, монолитная архитектура демонстрирует высокую производительность, но ограничена в масштабируемости, тогда как Domain-Driven Design подходит для сложных доменных моделей, но требует оптимизации межсервисных взаимодействий. Подобные выводы основаны на достоверных данных и могут служить ориентиром для дальнейших исследований в этой области.
Стиль изложения статьи научный, но при этом доступный для понимания. Структура работы логична и последовательна: от постановки проблемы и описания методологии до представления результатов и их анализа. Каждый раздел дополняет предыдущий, создавая целостную картину исследования. Использование иллюстраций и таблиц удачно подчеркивает ключевые моменты, а ссылки на авторитетные источники подтверждают глубину проработки темы.
Практическая значимость работы очевидна. Результаты исследования могут быть непосредственно применены при проектировании и оптимизации микросервисных систем. Автор не только выявляет преимущества и недостатки каждого подхода, но и

предлагает конкретные рекомендации, такие как необходимость оптимизации запросов к данным для Data-Driven Approach или тщательное планирование декомпозиции при переходе от монолита к микросервисам. Это делает статью полезной не только для теоретиков, но и для практикующих специалистов.

Интерес читательской аудитории к данной работе будет высоким. Материал представляет ценность для разработчиков, архитекторов и исследователей, занимающихся вопросами микросервисных архитектур. Четкие выводы и практические рекомендации позволяют использовать статью в качестве руководства при принятии проектных решений. Кроме того, предложенные направления для будущих исследований, такие как изучение асинхронных паттернов или влияние оркестрации Kubernetes, открывают новые перспективы для научных дискуссий.

В заключение следует отметить, что статья «Анализ гранулярности микросервисов: эффективность архитектурных подходов» является качественным и завершенным исследованием, сочетающим теоретическую глубину с практической применимостью. Работа выполнена на высоком академическом уровне, методологически обоснована и актуальна. Учитывая значительный вклад в область микросервисных архитектур, статью рекомендуется принять к публикации без доработок.