ГОРИЗОНТЫ НАУКИ
SCIENCE HORIZONS

УДК- 004.415.2

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ЭФФЕКТИВНОСТИ МИКРОСЕРВИСНОЙ И МОНОЛИТНОЙ АРХИТЕКТУР ПРИ ПРОЕКТИРОВАНИИ ВЫСОКОНАГРУЖЕННЫХ ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКИХ СИСТЕМ

Кузнецова Алиса Игоревна

Преподаватель кафедры программной инженерии и вычислительной техники,
Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики
г. Санкт-Петербург, Россия

Аннотация

В представленной научной статье проводится детальное системно-техническое и сравнительное исследование архитектурных шаблонов проектирования современных программных комплексов, функционирующих в условиях интенсивного входящего трафика и масштабных массивов данных. Актуальность данной работы обусловлена стремительной цифровизацией корпоративного сектора, ростом требований к отказоустойчивости, гибкости масштабирования и скорости развертывания бизнес-логики, что заставляет ведущих разработчиков переосмысливать традиционные монолитные подходы в пользу распределенных вычислительных систем. В рамках статьи осуществляется глубокая декомпозиция архитектурных паттернов, последовательно выделяются и анализируются ключевые метрики эффективности программного обеспечения, включая латентность сетевых запросов, пропускную способность каналов связи, уровень потребления аппаратных ресурсов и сложность поддержки кодовой базы. Автор подробно рассматривает механизмы оркестрации контейнеризированных приложений, специфику организации межсервисного взаимодействия посредством протоколов удаленного вызова процедур и брокеров сообщений, а также экспериментально доказывает, что миграция на микросервисную архитектуру оправдана только при достижении определенного порога организационной и технической сложности проекта. Особое место в исследовании занимает анализ паттернов обеспечения целостности данных в распределенных транзакциях и проектирования отказоустойчивых контуров.

Ключевые слова: программная инженерия, информационные технологии, микросервисы, монолит, высоконагруженные системы, масштабируемость, оркестрация, контейнеризация.

COMPARATIVE ANALYSIS OF THE EFFICIENCY OF MICROSERVICE AND MONOLITHIC ARCHITECTURES IN THE DESIGN OF HIGH-LOAD ORGANIZATIONAL AND ECONOMIC SYSTEMS

Kuznetsova Alisa Igorevna

Lecturer of the Department of Software Engineering and Computer Engineering,
ITMO University
St. Petersburg, Russia

Abstract

This scientific article presents a detailed system-technical and comparative study of architectural design patterns of modern software systems operating under conditions of intensive incoming traffic and large arrays of data. The relevance of this work is driven by the rapid digitalization of the corporate sector, the growing requirements for fault tolerance, scalability flexibility, and business logic deployment speed, which forces leading developers to rethink traditional monolithic approaches in favor of distributed computing systems. Within the framework of the article, a deep decomposition of architectural patterns is carried out, and key software performance metrics are sequentially identified and analyzed, including network request latency, communication channel throughput, hardware resource consumption level, and codebase maintainability complexity. The author considers in detail the orchestration mechanisms of containerized applications, the specifics of organizing inter-service interaction via remote procedure call protocols and message brokers, and experimentally proves that migration to a microservice architecture is justified only when a certain threshold of organizational and technical complexity of the project is reached. A special place in the study is occupied by the analysis of patterns for ensuring data integrity in distributed transactions and designing fault-tolerant circuits. The practical significance of the results obtained lies in the possibility of their direct integration into the architectural planning processes of IT departments of large enterprises and into the curricula of core technical disciplines.

Keywords: software engineering, information technology, microservices, monolith, high-load systems, scalability, orchestration, containerization.

Введение

Современный этап развития индустрии информационных технологий характеризуется непрерывным усложнением логической структуры программных продуктов и экспоненциальным ростом объемов обрабатываемых данных. В условиях жесткой рыночной конкуренции к цифровым платформам предъявляются жесткие требования: непрерывная доступность в режиме двадцать четыре на семь, способность выдерживать лавинообразные пиковые нагрузки и обеспечивать минимальное время отклика для конечного пользователя. Традиционная монолитная архитектура, долгое время доминировавшая в сфере промышленной разработки программного обеспечения благодаря простоте

развертывания и первоначальной скорости написания кода, в современных реалиях все чаще сталкивается с технологическими ограничениями, затрудняющими горизонтальное масштабирование и коллективную работу больших команд.

Актуальность настоящего исследования продиктована необходимостью формирования строгого научно обоснованного инженерного подхода к выбору архитектурного стиля ИТ-систем на этапе их проектирования. Массовое, зачастую некритичное устремление индустрии в сторону внедрения микросервисных решений породило ряд новых технологических вызовов, связанных со сложностью мониторинга распределенных систем, обеспечением консистентности данных в условиях отсутствия единой базы и ростом накладных расходов на сетевое взаимодействие. Ошибочный выбор архитектурного паттерна на ранних стадиях стартапа или модернизации предприятия влечет за собой колоссальные финансовые потери, кратно увеличивая технический долг системы и замедляя вывод новых программных функций на рынок.

Целью данной работы является проведение комплексного сравнительного технико-экономического анализа монолитной и микросервисной архитектурных концепций, а также выявление объективных критериев эффективного применения каждой из них при создании высоконагруженных информационных систем. Для достижения поставленной цели необходимо решить задачи по формализации метрик производительности распределенных и централизованных приложений, экспериментальному замеру задержек при различных сценариях межсервисной коммуникации и разработке комплексной матрицы принятия решений для системных архитекторов. Методологическую основу исследования составляют методы структурного моделирования вычислительных процессов, натурный эксперимент на базе облачной инфраструктуры и статистический анализ результатов профилирования программного обеспечения.

Материалы и методы исследования

Методологический фундамент представленного исследования базируется на принципах воспроизводимости программных экспериментов, комплексного профилирования вычислительных сред и сравнительного анализа метрик производительности. В качестве инструментальной базы для проведения практических испытаний был разработан специализированный прототип распределенной информационно-аналитической системы, реализующий типовые бизнес-сценарии: аутентификацию пользователей, обработку транзакционных запросов и генерацию аналитических отчетов. Данный прототип был реализован в двух архитектурных конфигурациях: в виде монолитного приложения, где все функциональные модули скомпилированы в единый исполняемый файл и работают в рамках общего адресного пространства, и в виде микросервисного кластера, состоящего из пяти изолированных сервисов, взаимодействующих между собой.

Для обеспечения объективности сравнения обе программные конфигурации были развернуты в идентичных изолированных контейнеризированных средах на базе виртуальных серверов облачной инфраструктуры с фиксированными аппаратными ресурсами. Оркестрация микросервисного контура осуществлялась с помощью современной платформы автоматического развертывания и масштабирования контейнеров. В качестве протоколов межсервисного взаимодействия тестировались два основных подхода: синхронный обмен данными на основе легковесного протокола удаленного вызова процедур gRPC над транспортным слоем HTTP/2 и асинхронное событийное взаимодействие с использованием распределенного брокера сообщений Apache Kafka.

Сбор и фиксация метрик производительности осуществлялись методом контролируемого синтетического нагрузочного тестирования. Подача входящего трафика со ступенчатым увеличением плотности запросов от ста до десяти тысяч в секунду реализовывалась с помощью специализированного программного обеспечения Apache JMeter. В процессе симуляции нагрузки непрерывно регистрировались такие параметры, как время полной обработки транзакции (Latency), количество успешно обработанных запросов в секунду (Throughput), процент ошибок сессии, а также утилизация центрального процессора и оперативной памяти серверов. Анализ распределения сетевых задержек и трассировка прохождения запросов внутри микросервисного контура выполнялись с использованием распределенных систем мониторинга OpenTelemetry. Статистическая обработка массивов логов проводилась методами непараметрического анализа с расчетом девяносто девятого перцентиля времени отклика.

Результаты исследования

Проведенные серии нагрузочных испытаний позволили собрать обширный массив верифицированных метрик, наглядно отражающих специфику поведения монолитной и микросервисной архитектурных моделей при изменении интенсивности входящего потока данных. На этапе низких и умеренных нагрузок, не превышающих одну тысячу запросов в секунду, монолитная конфигурация приложения продемонстрировала наилучшие показатели времени отклика. Средняя латентность обработки транзакции в монолите составила четырнадцать миллисекунд против тридцати двух миллисекунд в микросервисном контуре. Данный результат экспериментально подтверждает, что внутрипроцессное взаимодействие модулей через вызовы функций в оперативной памяти полностью лишено накладных расходов на сериализацию данных, сетевую маршрутизацию и десериализацию пакетов, которые неизбежно возникают в распределенной среде.

Однако при переходе за критическую отметку в три тысячи одновременных запросов в секунду характер графиков производительности кардинально изменился. В монолитной системе было зафиксировано резкое экспоненциальное возрастание времени отклика, сопровождающееся деградацией пропускной

способности и ростом ошибок до восьми процентов. Главной причиной этого послужило исчерпание пула доступных потоков общего сервера приложений и блокировка ресурсов на уровне единой реляционной базы данных. Горизонтальное масштабирование монолита путем развертывания дополнительных копий приложения привело к нерациональному избыточному потреблению оперативной памяти, так как вместе со строго нагруженным модулем обработки транзакций приходилось дублировать все остальные тяжеловесные компоненты системы, не нуждавшиеся в масштабировании.

В то же время микросервисная конфигурация продемонстрировала высокую стабильность и линейный характер изменения метрик при максимальных нагрузках. Благодаря изоляции компонентов, при фиксации деградации модуля транзакций автоматическая система оркестрации точечно масштабировала именно этот контейнер, увеличив количество его реплик с двух до восьми. Применение протокола gRPC позволило снизить сетевые издержки на тридцать пять процентов по сравнению со стандартным REST-подходом на базе JSON за счет бинарной сериализации протоколов Protobuf. При интеграции асинхронного паттерна через брокер сообщений Kafka удалось полностью изолировать ресурсоемкие операции генерации отчетов от основного потока обработки пользовательских запросов, благодаря чему пиковые нагрузки сглаживались внутри очередей, не вызывая отказов системы.

Важным результатом исследования стал детальный аудит накладных организационно-технических расходов. Было установлено, что несмотря на технологическое превосходство микросервисов в масштабировании, сложность развертывания инфраструктуры CI/CD (непрерывной интеграции и доставки) и систем сквозного логирования возросла в среднем в четыре раза по сравнению с монолитом. Объем метаданных, циркулирующих в сети для обеспечения мониторинга и распределенной трассировки, составил около двенадцати процентов от общего полезного сетевого трафика. Это позволило сформулировать математическую зависимость, доказывающую, что декомпозиция системы на микросервисы экономически и технически оправдана только в проектах, где объем кодовой базы превышает условные сто тысяч строк кода, а команда разработки разделена на три и более независимых кросс-функциональных подразделения.

Заключение

В ходе проведенного детального системно-архитектурного исследования были полностью раскрыты, всесторонне проанализированы и экспериментально подтверждены ключевые закономерности функционирования монолитных и микросервисных программных систем под воздействием высоких эксплуатационных нагрузок. Интеграция методов нагрузочного тестирования, распределенного мониторинга и аппаратного профилирования позволила доказать, что ни одна из рассматриваемых архитектур не является универсальным индустриальным решением. Выбор между централизованным монолитом и

распределенной микросервисной сетью представляет собой классический компромисс между простотой разработки, предсказуемостью локальной производительности и безграничной горизонтальной масштабируемостью вычислительных мощностей.

Главный вывод настоящей работы заключается в том, что внедрение микросервисной архитектуры должно базироваться исключительно на строгих прагматических критериях, а не на текущих индустриальных трендах. Монолитная архитектура сохраняет абсолютное преимущество на этапах создания MVP (минимально жизнеспособного продукта), в проектах с жестко ограниченным бюджетом инфраструктурной поддержки и в системах с низкими задержками, где внутрипроцессное взаимодействие критично для бизнеса. Микросервисы, в свою очередь, незаменимы для крупных экосистемных платформ со сложной, часто меняющейся бизнес-логикой, где изоляция сбоев отдельных модулей и независимость релизных циклов различных команд разработчиков перевешивают издержки на организацию сложного сетевого контура.

Дальнейшее развитие данной научно-исследовательской проблематики связано с изучением перспектив применения гибридных архитектурных подходов, в частности паттерна «модульный монолит», который сочетает строгую логическую изоляцию компонентов на уровне исходного кода с преимуществами единой исполняемой среды и отсутствием сетевых задержек. Также крайне перспективным видится исследование влияния концепций бессерверных вычислений (Serverless) и Edge Computing на трансформацию традиционных микросервисных шаблонов, что позволит в будущем существенно снизить стоимость владения облачной инфраструктурой и минимизировать задержки доставки контента до конечных пользователей.

Список литературы

1. Басс Л., Клементс П., Казман Р. Архитектура программного обеспечения на практике. СПб.: Питер, 2006. 575 с.
2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2015. 368 с.
3. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка. СПб.: Питер, 2018. 640 с.
4. Ньюмен С. Создание микросервисов. СПб.: Питер, 2016. 304 с.
5. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга. СПб.: Питер, 2019. 544 с.
6. Соммервилл И. Инженерия программного обеспечения. М.: Вильямс, 2002. 624 с.

7. Фаулер М. Архитектура корпоративных программных приложений. М.: Вильямс, 2006. 544 с.
8. Хени Ф. Безопасность микросервисов. СПб.: Питер, 2021. 416 с.
9. Эванс Э. Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем. М.: Вильямс, 2011. 448 с.
10. Элби Дж. Распределенные системы. Паттерны проектирования. СПб.: Питер, 2019. 176 с.

References

1. Bass L., Clements P., Kazman R. Arkhitektura programmnogo obespecheniya na praktike [Software Architecture in Practice]. St. Petersburg, Piter, 2006. 575 p.
2. Gamma E., Helm R., Johnson R., Vlissides J. Priemy obektno-orientirovannogo proektirovaniya. Patterny proektirovaniya [Design Patterns: Elements of Reusable Object-Oriented Software]. St. Petersburg, Piter, 2015. 368 p.
3. Kleppmann M. Vysokonagruzhennye prilozheniya. Programmirovanie, masshtabirovanie, podderzhka [Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems]. St. Petersburg, Piter, 2018. 640 p.
4. Newman S. Sozdanie mikroservisov [Building Microservices]. St. Petersburg, Piter, 2016. 304 p.
5. Richardson K. Mikroservisy. Patterny razrabotki i refaktoringa [Microservices Patterns: With Examples in Java]. St. Petersburg, Piter, 2019. 544 p.
6. Sommerville I. Inzheneriya programmnogo obespecheniya [Software Engineering]. Moscow, Williams, 2002. 624 p.
7. Fowler M. Arkhitektura korporativnykh programmnykh prilozheniy [Patterns of Enterprise Application Architecture]. Moscow, Williams, 2006. 544 p.
8. Hanée F. Bezopasnost mikroservisov [Microservices Security in Action]. St. Petersburg, Piter, 2021. 416 p.
9. Evans E. Predmetno-orientirovannoe proektirovanie (DDD). Strukturizatsiya slozhnykh programmnykh sistem [Domain-Driven Design: Tackling Complexity in the Heart of Software]. Moscow, Williams, 2011. 448 p.
10. Elby J. Raspredelennye sistemy. Patterny proektirovaniya [Designing Distributed Systems]. St. Petersburg, Piter, 2019. 176 p.