УДК 004.75

МЕТОДИКА ОЦЕНКИ ПРОИЗВОДИТЕЛЬНОСТИ МИКРОСЕРВИСНОГО ВЗАИМОДЕЙСТВИЯ В ВЕБ-ФРЕЙМВОРКАХ

Моничев И. Д.[1], Черяев А. С.[1]
Научный руководитель – кандидат педагогических наук, доцент Государев И. Б.[1]
[1]Университет ИТМО
id.monichev@gmail.com, sasha10131310@gmail.com

Введение

Современные веб-приложения всё чаще строятся на базе микросервисной архитектуры, которая обеспечивает независимое развитие и масштабирование компонентов системы. При проектировании таких решений возникает необходимость в достоверной и сопоставимой оценке производительности серверных веб-фреймворков.

Вопрос заключается в отсутствии единого подхода к оценке производительности микросервисного взаимодействия. Различия в архитектуре тестовых приложений, сценариях нагрузки и условиях проведения экспериментов приводят к тому, что полученные результаты трудно сопоставлять и использовать при выборе технологического стека [1].

В работе предложена унифицированная методика оценки производительности микросервисного взаимодействия, основанная на единой архитектурной модели и контролируемом нагрузочном тестировании. Методика проверена на веб-фреймворках NestJS (JavaScript) [2], ASP.NET Core (C#) [3] и Gin (Golang) [4].

Основная часть

В качестве предметной области рассматривается торговая платформа, включающая базовые операции между пользователями, заказами и платежами. Архитектура реализована с применением шаблона проектирования API Gateway, через который выполняется синхронное взаимодействие между микросервисами [5]. Асинхронное взаимодействие реализовано с использованием брокера сообщений.

Для обеспечения сопоставимости результатов во всех реализациях сохранены одинаковые:

- бизнес-логика и структура взаимодействия сервисов;
- используемые базы данных и начальное состояние данных;
- параметры окружения и ресурсы выполнения;
- сценарии нагрузки.

Оценка производительности проводилась по следующим метрикам:

- среднее время отклика и p95;
- пропускная способность (RPS);
- загрузка CPU;
- потребление оперативной памяти.

Экспериментальное окружение было изолировано за счёт контейнеризации с использованием Docker. Применялся метод нагрузочного тестирования с применением инструментов k6 [6], Prometheus [7] и cAdvisor [8]. Использовались типовые сценарии взаимодействия: получение одного объекта, получение списка объектов, создание нового объекта и асинхронное обновление статуса через механизм обмена сообщениями. Каждый сценарий выполнялся многократно, после чего полученные значения метрик усреднялись для повышения статистической достоверности результатов.

Выводы

Результаты показали, что реализации на Gin демонстрируют минимальное время отклика и наименьшее потребление ресурсов. Веб-фреймворк ASP.NET Core

обеспечивает сопоставимую стабильность отклика при более высоком потреблении памяти. NestJS показывает более высокие задержки и загрузку процессора, что связано с особенностями среды выполнения JavaScript, однако сохраняет приемлемую пропускную способность.

Разработанная методика сравнительной оценки производительности микросервисного взаимодействия, обеспечивает воспроизводимость и сопоставимость результатов за счёт единой архитектуры, изолированного окружения и контролируемой нагрузки.

Практическая значимость работы заключается в возможности применения предложенного подхода при выборе веб-фреймворка для микросервисных систем.

Перспективы дальнейших исследований связаны с расширением набора сценариев, а также анализом влияния используемых СУБД, ORM и механизмов сериализации на производительность.

Литература

1. Ponce F., Verdecchia R., Miranda B., Soldani J. Microservices testing: A systematic literature review // Information and Software Technology. 2025. Vol. 188.
2. NestJS Documentation [Электронный ресурс]. – Режим доступа: https://docs.nestjs.com (дата обращения: 18.02.2026).
3. ASP.NET Core Documentation [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/aspnet/core (дата обращения: 18.02.2026).
4. Gin Web Framework Documentation [Электронный ресурс]. – Режим доступа: https://gin-gonic.com/en/docs (дата обращения: 18.02.2026).
5. Zuo X., Su Y., Wang Q., Xie Y. An API gateway design strategy optimized for persistence and coupling // Advances in Engineering Software. 2020. Vol. 148.
6. Chandrasekhar A. K., Chandran A. S. Comparative Analysis of Load Testing Tools // International Journal of Creative Research Thoughts (IJCRT). 2021. Vol. 9, Issue 6. ISSN 2320-2882.
7. Prometheus Documentation [Электронный ресурс]. – Режим доступа: https://prometheus.io/docs (дата обращения: 18.02.2026).
8. cAdvisor Documentation [Электронный ресурс]. – Режим доступа: https://github.com/google/cadvisor (дата обращения: 18.02.2026).