UDC 004.415 **doi: 10.26906/SUNZ.2026.1.177**

Oleksandr Vasylchenko, Anton Poroshenko, Roman Yaroshevych, Vladislav Kholiev, Maksym Shostak

Kharkiv National University of Radio Electronics, Kharkiv, Ukraine

MODEL FOR ORGANIZING HYBRID COMMUNICATION IN A MICROSERVICE ARCHITECTURE ON THE .NET PLATFORM

Abstract. Relevance. Modern microservice systems are widely used in high-load software products, particularly in cloud and enterprise environments, where performance, scalability, and reliability are critical requirements. The choice of interaction mechanisms between microservices significantly affects system behavior under load and its ability to handle peak request spikes. **Object of research.** The object of research is the interaction processes between microservices in distributed applications on the .NET platform. **Purpose of the article.** The purpose of the article is to analyze synchronous and asynchronous communication models, experimentally compare their performance under different load conditions, and substantiate a hybrid interaction model that combines the advantages of both approaches. **Research results.** Experimental performance testing of synchronous and asynchronous processing models was conducted. The results demonstrate that asynchronous interaction provides better scalability and resilience under high load, while the synchronous model is more effective for moderate traffic and low-latency scenarios. A hybrid model is proposed that dynamically selects the interaction type based on an integral load assessment. **Conclusions.** It is concluded that the use of a hybrid model enables stable and efficient operation of microservice systems by adaptively switching between synchronous and asynchronous processing depending on the current system state.

Keywords: microservice architecture, .NET, synchronous interaction, asynchronous interaction, performance, hybrid model.

Introduction

Current State. Modern distributed systems are increasingly used in large software products with high requirements for performance, scalability, and reliability. Microservice architecture enables flexible decomposition of monolithic applications into independent services, which simplifies the development, updating, and deployment of individual components. However, as such systems evolve, their structure becomes more complex and the volume of load increases, which intensifies the problem of choosing an correct method of interaction between services. An incorrect decision regarding the use of synchronous or asynchronous communication mechanisms, as well as coordination approaches, can lead to increased latency, the emergence of bottlenecks, and difficulties in system maintenance.

Modern microservice systems are widely used in high-load software products, particularly in cloud and enterprise environments, where scalability, fault tolerance, and predictable performance are key requirements. In the practice of designing such systems, the primary focus is placed not only on functional decomposition but also on the choice of interaction mechanisms between services, which significantly affects system behavior underload [1]. In most modern implementations, both synchronous and asynchronous communication mechanisms are used. Synchronous interaction, typically implemented through HTTP calls, remains dominant for latency-sensitive operations due to its simplicity of implementation and transparency of request processing. At the same time, practical experience in operating large-scale microservice systems shows that as load increases, synchronous calls become a source of blocking and cascading failures, which limits system scalability [2].

Asynchronous approaches based on message exchange and event-driven architecture are increasingly applied to the processing of bulk and background operations. They allow load leveling, reduce coupling between services, and improve resilience to peak request spikes. At the same time, extensive use of asynchronous interaction complicates system logic, requires additional infrastructure, and increases the requirements for monitoring, error handling, and ensuring idempotency [3]. In the .NET ecosystem, these trade-offs are particularly pronounced, as the platform is actively used to build both high-performance APIs and distributed services with complex business logic. Practical guidelines for developing .NET microservices usually recommend fixing the interaction model at the design stage; however, such an approach does not account for the dynamic nature of load in real-world systems [4].

Problem Statement. The objective of this work is to study and improve approaches to organizing microservice applications on the .NET platform. The study is aimed at identifying correct interaction strategies with consideration of the specifics of the .NET environment, experimentally comparing synchronous and asynchronous models under different load conditions, and substantiating a hybrid model in which both approaches are combined to achieve stable and efficient system operation. This will enable developers to obtain recommendations for the design, implementation, and scaling of microservice systems that meet modern requirements for performance, reliability, and efficiency.

Main Section

Study of Interaction Methods for Microservice Applications. The way in which microservices interact with each other directly affects the performance, scalability, and reliability of the system [5]. The choice of message exchange approach determines how flexibly the system responds to peak loads, how quickly it can recover from failures, and how easily it can adapt to new requirements.

Synchronous communication has several advantages. It is characterized by straightforward integration with existing .NET frameworks such as ASP.NET, which simplifies the development and maintenance of services. In addition, the client receives a response immediately after the request is processed, which significantly simplifies further handling of the results. At the same time, this approach also has a few drawbacks. Synchronous interaction directly depends on the availability of the target service: if it is unavailable, request execution is blocked, which can lead to failure of the entire business process. Synchronous calls also limit scalability, since many concurrent requests can overload the service and significantly degrade performance. Moreover, in the case of long-running operations, the overall response time increases. To implement synchronous interaction in C#, HTTP/REST APIs using HttpClient or gRPC technology are typically used; the latter provides high-performance remote procedure calls based on the HTTP protocol [6].

Asynchronous message delivery allows a client to send a request without the need to wait for an immediate response, since requests and the results of their processing proceed independently of each other [7]. This approach ensures scalability by enabling the system to efficiently handle a significantly larger number of requests without the risk of overloading individual services. It also improves reliability, as the client does not depend on the immediate availability of a service, and reduces user-perceived latency, allowing the user to continue working without waiting for the operation to be completed. However, asynchronous interaction complicates implementation, requires additional mechanisms for handling responses and tracking request states, makes debugging more difficult due to the distributed nature of data flows, and also necessitates the implementation of idempotency to avoid duplicate operations when messages are resent. In C#, asynchronous scenarios are implemented using message queues, including Azure Service Bus, RabbitMQ, or Apache Kafka, as well as through event-driven architecture, where events serve as triggers for subsequent actions across different services.

During the study, a performance comparison of synchronous and asynchronous request processing models was conducted based on two different approaches. The synchronous model was implemented using the HttpClient class, which is the traditional method for performing HTTP requests in C#. At the same time, the asynchronous model was based on message exchange using RabbitMQ.

Testing was carried out under two different scenarios:

1. Dependence of execution time on the total number of requests.

2. Dependence of execution time on the number of requests per thread.

The first scenario (Fig. 1) demonstrated a clear difference between the two approaches. For the synchronous model, the execution time initially increased linearly with the growth in the number of requests. However, after reaching a critical threshold of 10^4 requests, the system exhausted its capacity, and the increase in execution time became exponential. This behavior is typical for synchronous execution, where threads are blocked while waiting for responses.

The asynchronous model demonstrated significantly more stable results. An increase in the number of requests led to linear growth in execution time, while the system efficiently handled even high loads without substantial delays. This can be explained by the capabilities of asynchronous processing, where tasks are executed in parallel and resources are used correctly through RabbitMQ queues.

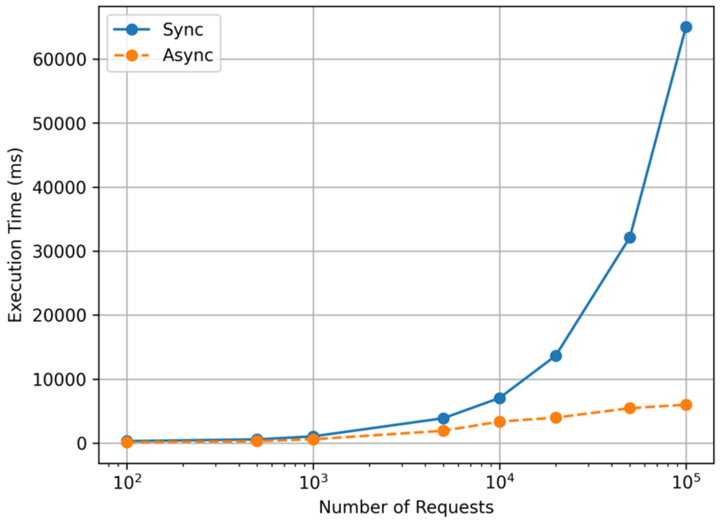

Fig. 1. Results of the first test

The second scenario (Fig. 2), which examined the impact of requests per thread, also confirmed the advantage of the asynchronous model. In the synchronous model, execution time began to increase significantly even at moderate values such as 25/2000. Further increases in the requests per thread resulted in overload and a substantial rise in execution time.

The asynchronous model coped with such load more effectively. Although execution time gradually increased, it remained significantly lower compared to the synchronous model. Only at very high levels, such as 75/1000 or 100/1000, did a gradual increase in execution time begin to appear, indicating that the system was approaching its performance limit.

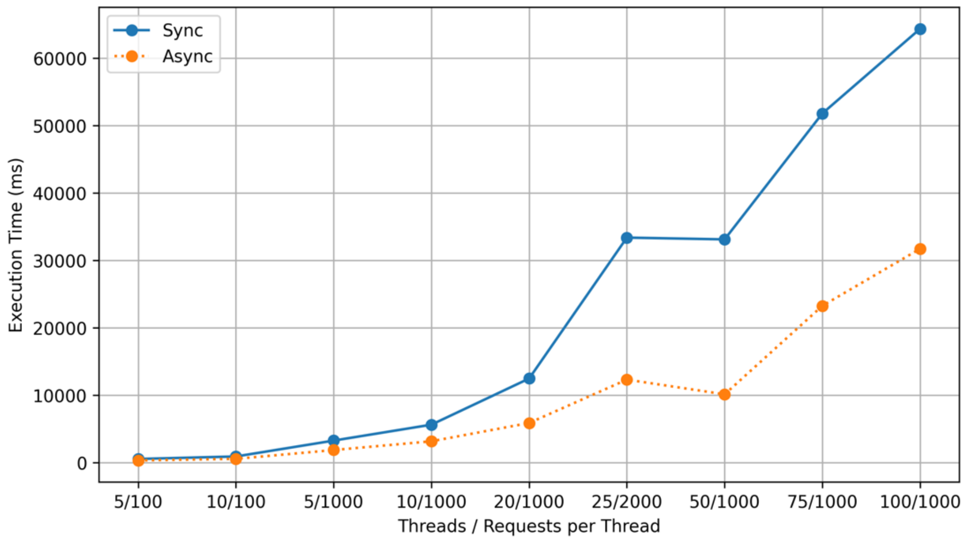

Fig. 2. Results of the second test

The results obtained demonstrate the advantage of the asynchronous message-passing model for processing many requests. The asynchronous approach enables more efficient use of system resources through parallel processing and message queues, whereas the synchronous model quickly reaches its performance limits. At the same time, the use of asynchronous communication is associated with the need to maintain additional infrastructure, including message brokers, and with increased operational costs in cloud environments, where charges are applied based on the number of processed messages and the volume of traffic. This confirms that neither a purely synchronous nor a purely asynchronous model is a universal solution [8].

Hybrid Model of Interaction Between Microservices. Considering the obtained results, a need arises to formulate a hybrid model of interaction between microservices that combines the advantages of both models. The idea is that the system does not fix the interaction type in advance but dynamically chooses between synchronous and asynchronous communication depending on the current load, latency requirements, and economic constraints. For example, during periods of low load, requests can be processed synchronously, which ensures minimal response time and does not require intensive use of brokers. During peak activity, part of the operations is shifted to the asynchronous mode, which helps avoid service overload and level the load on the infrastructure. Fig. 3 presents a conceptual diagram of the hybrid interaction model, which reflects the sequence of request processing in a microservice system. An incoming request is sent to the decision-making module, which determines the interaction mode based on operational metrics received from the monitoring module. Depending on the current load, either the synchronous or asynchronous mode is selected, after which the request is forwarded to the corresponding microservices. The processing result is produced by the microservices and returned to the client as a response.

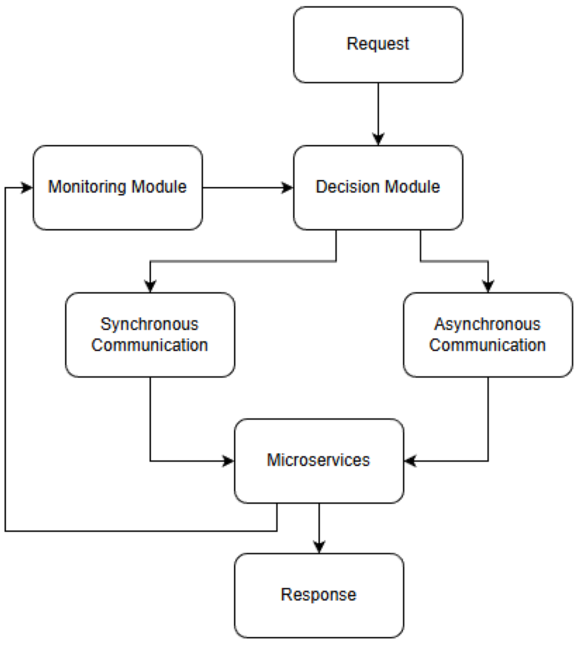

Fig. 3. Conceptual diagram of the hybrid model

To build the hybrid model, a set of parameters is used that quantitatively characterize the load and reflect the current state of the system. These parameters include the incoming request rate (RPS), the average service response time, and CPU utilization. Each of them has defined threshold values that specify the limits of stable operation. If the parameter values exceed these thresholds, the system approaches a degradation state, and under such conditions it is appropriate to switch to the asynchronous processing model. Operation within the lower thresholds, on the contrary, ensures the effectiveness of synchronous interaction.

To generalize these criteria, an integral load assessment is used:

$$L = \alpha \cdot \frac{RPS}{R_{high}} + \beta \cdot \frac{T_{resp}}{T_{high}} + \gamma \cdot \frac{CPU_{util}}{CPU_{high}},$$

where the coefficients α, β and γ determine the weight of each parameter. The obtained value allows decisions to be made in a unified manner:

$$sync: L < 1,$$

$$async: L \geq 1.$$

This approach makes it possible to evaluate the load comprehensively and ensures more stable and predictable switching between the sync and async modes. Several implementation options for the hybrid model in a microservice system on the .NET platform can be distinguished:

- hybrid at the API Gateway level. Incoming requests arrive at the gateway, where the decision-making module is implemented.
- hybrid at the individual service level. The entry service always accepts the request synchronously, but different processing paths are used inside it.
- hybrid at the individual operation level. For each endpoint or business action, an important class is defined: latency-critical, cost-sensitive, background processing, and so on.

Unlike classical approaches, where the choice of synchronous or asynchronous communication is made at the design stage and remains unchanged during operation, this model treats the interaction type as a dynamic parameter that changes over time.

Conclusions

This paper considered two main approaches to interaction between microservices: synchronous and asynchronous. It was shown that the synchronous model is simpler to understand and implement and is well suited for scenarios with moderate load and strict response-time requirements. In contrast, the asynchronous model performs better under high-load conditions, providing improved scalability and resilience to peak request spikes. Experimental results confirmed that as the request volume increases, synchronous interaction quickly reaches its limits due to the blocking nature of processing, whereas the asynchronous model allows more uniform utilization of system resources. At the same time, it was established that pervasive use of asynchronous interaction leads to increased system complexity and higher operational costs associated with maintaining additional message-exchange infrastructure, especially in cloud environments with pay-per-use pricing.

Based on these findings, it was concluded that neither a purely synchronous nor a purely asynchronous approach is universal. The application of a hybrid model is therefore appropriate, in which the interaction type is selected depending on the current load and economic constraints. In simpler and less loaded scenarios, the system may predominantly use synchronous interaction, while upon reaching critical load or latency thresholds, part of the operations can be shifted to the asynchronous mode, thereby leveling the load on the infrastructure.

Within the scope of this work, the hybrid model is described at a conceptual level without proceeding to full implementation. The metrics used and the operating principles of the decision-making module serve as a basis for further practical research and testing of dynamic selection between synchronous and asynchronous interaction modes. Thus, the following results were obtained in this article:

1. A review of synchronous and asynchronous interaction models in .NET microservice systems was conducted, and their advantages, disadvantages, and conditions for effective application were analyzed.
2. Experimental performance testing of both models was carried out under two load regimes, and the results demonstrated differences in scalability and system behavior under high incoming request intensity and multithreading.
3. A conceptual hybrid interaction model was proposed, in which the request processing type is determined based on an integral load assessment. This approach provides more stable switching between synchronous and asynchronous processing and improves overall system efficiency.

Future research in the field of microservice interaction may focus on further elaboration of the practical aspects of implementing the proposed approach, including the development of a full-featured prototype and its integration into a .NET microservice environment. Particular attention may be given to extended load testing and analysis of model behavior under various traffic types, service configurations, and scaling scenarios, which will make it possible to assess the robustness and applicability of the model in real-world systems.

Конфлікт інтересів

Автори декларують, що не мають конфлікту інтересів стосовно даного дослідження, в тому числі фінансового, особистісного характеру, авторства чи іншого характеру, що міг би вплинути на дослідження та його результати, представлені в даній статті.

Використання засобів штучного інтелекту

Автори підтверджують, що не використовували технології штучного інтелекту при створенні представленої роботи.

Reference

1. Newman S. Building Microservices. O'Reilly Media, Incorporated, 2015. URL: https://www.oreilly.com/library/view/building-microservices/9781491950340
2. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud / M. Villamizar et al. *2015 10th Computing Colombian Conference (10CCC)*, Bogota, Colombia, 21–25 September 2015. 2015. URL: https://doi.org/10.1109/columbiancc.2015.7333476

3. Microservices: Yesterday, Today, and Tomorrow / N. Dragoni et al. *Present and Ulterior Software Engineering*. Cham, 2017. P. 195–216. URL: https://doi.org/10.1007/978-3-319-67425-4_12
4. Pahl C. Containerization and the PaaS Cloud. *IEEE Cloud Computing*. 2015. Vol. 2, no. 3. P. 24–31. URL: https://doi.org/10.1109/mcc.2015.51
5. Pinciroli R., Aleti A., Trubiani C. Performance Modeling and Analysis of Design Patterns for Microservice Systems. *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, L'Aquila, Italy, 13–17 March 2023. 2023. URL: https://doi.org/10.1109/icsa56044.2023.00012
6. Microservices: Migration of a Mission Critical System / M. Mazzara et al. *IEEE Transactions on Services Computing*. 2018. P. 1. URL: https://doi.org/10.1109/tsc.2018.2889087
7. Gordesli M., Varol A. Comparing Interservice Communications of Microservices for E-Commerce Industry. 2022 10th International Symposium on Digital Forensics and Security (ISDFS), Istanbul, Turkey, 6–7 June 2022. 2022. URL: https://doi.org/10.1109/isdfs55398.2022.9800784
8. Analyzing Microservices and Monolithic Systems: Key Factors in Architecture, Development, and Operations / J. Christian et al. *2023 6th International Conference of Computer and Informatics Engineering*, Lombok, Indonesia, 14–15 September. 2023. URL: https://doi.org/10.1109/ic2ie60547.2023.10331155

Received (Надійшла) 18.12.2025
Accepted for publication (Прийнята до друку) 11.02.2026
Publication date (Дата публікації) 27.02.2026

Відомості про авторів / About the Authors

Васильченко Олександр Сергіович – студент кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Vasylchenko Oleksandr – student, Department of Electronic Computers, Kharkiv National University of Radio Electronics Kharkiv, Ukraine;
e-mail: oleksandr.vasylchenko@nure.ua; ORCID Author ID: https://orcid.org/0009-0007-9265-7825.

Порошенко Антон Ігорович – доктор філософії, старший викладач кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Anton Poroshenko – PhD, Senior Lecturer, Department of Electronic Computing Machines, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine;
e-mail: anton.poroshenko@nure.ua; ORCID Author ID: https://orcid.org/0000-0001-7266-4269;
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=57250025600.

Ярошевич Роман Олександрович – доктор філософії, старший викладач кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Roman Yaroshevych – PhD, Senior Lecturer, Department of Electronic Computers, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine;
e-mail: roman.yaroshevych@nure.ua; ORCID Author ID: https://orcid.org/0000-0002-7949-1513;
Scopus Author ID https://www.scopus.com/authid/detail.uri?authorId=58624172500.

Холєв Владислав Олександрович – доктор філософії, асистент кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Vladyslav Kholiev – PhD, Assistant Professor, Department of Electronic Computing Machines, National University of Radio Electronics, Kharkiv, Ukraine;
e-mail: vladyslav.kholiev@nure.ua; ORCID Author ID: https://orcid.org/0000-0002-9148-1561;
Scopus Author ID https://www.scopus.com/authid/detail.uri?authorId=57224189723.

Шостак Максим Віталійович – аспірант кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Maksym Shostak – PhD student, Department of Electronic Computers, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine;
e-mail: maksym.shostak@nure.ua; ORCID Author ID: http://orcid.org/0009-0003-1510-7425.

Модель організації гібридної комунікації в мікросервісній архітектурі на платформі .NET

О. С. Васильченко, А. І. Порошенко, Р. О. Ярошевич, В. О. Холєв, М. В. Шостак

Анотація. Актуальність. Сучасні мікросервісні системи широко використовуються у високонавантажених програмних продуктах, зокрема в хмарних та корпоративних середовищах, де ключовими вимогами є продуктивність, масштабованість і надійність. Вибір способу взаємодії між мікросервісами істотно впливає на поведінку системи під навантаженням і її здатність протистояти піковим сплескам запитів. **Об'єкт дослідження.** Об'єктом дослідження є процеси взаємодії між мікросервісами у розподілених додатках на платформі .NET. **Мета статті.** Метою статті є аналіз синхронних та асинхронних моделей комунікації, експериментальне порівняння їх продуктивності за різних режимів навантаження та обґрунтування гібридної моделі взаємодії, що поєднує переваги обох підходів. **Результати дослідження.** Проведено експериментальне тестування синхронної та асинхронної моделей обробки запитів, яке показало кращу масштабованість і стійкість асинхронної взаємодії за високих навантажень та ефективність синхронної моделі за помірного трафіку. Запропоновано гібридну модель, що динамічно обирає тип взаємодії на основі інтегральної оцінки навантаження. **Висновки.** Зроблено висновок, що застосування гібридної моделі дозволяє забезпечити стабільну та ефективну роботу мікросервісних систем шляхом адаптивного перемикання між синхронною та асинхронною обробкою залежно від поточного стану системи.

Ключові слова: мікросервісна архітектура, .NET, синхронна взаємодія, асинхронна взаємодія, продуктивність, гібридна модель.