Підвищення відмовостійкості мікросервісів високонавантажених систем на основі спостереження за навантаженням

Ігор Папроцький[1], *Анатолій Пашко*[2]

[1,2] Київський національний університет імені Тараса Шевченка,
вул. Володимирська, 64, м.Київ, Україна, 01033
[1] igorpapr@gmail.com, https://orcid.org/0009-0006-1644-2833,
[2] aapashko@univ.kiev.ua, https://orcid.org/0000-0001-6944-8477

Received 16.10.2025, accepted 11.12.2025
https://doi.org/10.32347/st.2025.4.1205

Анотація. В статті досліджується проблема налаштування комунікації між серверами високонавантажених систем, що функціонують в умовах періодичних навантажень. Основна увага приділяється шаблонам відмовостійкості, які безпосередньо впливають на синхронну комунікацію між сервісами. В фокусі цього дослідження метод Circuit Breaker, що формалізує аналогію із реальним вимикачем для контролю та ізоляції відмов у розподілених системах.

Мета роботи – підвищення відмовостійкості мікросервісу шляхом динамічного регулювання розміру ковзного вікна збору даних, що надає змогу адаптувати контекст до поточних коливань трафіку.

Для досягнення цієї мети розроблено модель обчислення цільового розміру вікна на основі аналізу навантаження в спостережуваному проміжку часу, яку покладено в основу роботи Circuit Breaker.

Описано алгоритм, що періодично обчислює і адаптує розмір ковзного вікна збору даних до коливань трафіку в системі, базуючись на обчисленнях і експоненційному згладжуванні поточного навантаження, які виконуються згідно з запропонованою моделлю.

Практичне значення отриманих результатів вбачається в можливості впровадження нових алгоритмів покращення рейтингу комунікації мікросервісів високонавантажених систем, що дозволить підвищити їх відмовостійкість і забезпечити стабільність роботи систем за різних умов навантаження. Наукова новизна полягає у подальшому розвитку методології підвищення відмовостійкості мікросервісів високонавантажених систем, що грунтується на визначені стану їх взаємодії за статистичними даними попередніх запитів і їх результатів.

Ключові слова: адаптивність, ковзне вікно, синхронна комунікація, трафік, Circuit Breaker, шаблон відмовостійкості.

Ігор Папроцький
аспірант кафедри теоретичної кібернетики Київського національного університету імені Тараса Шевченка

Анатолій Пашко
Професор кафедри теоретичної кібернетики Київського національного університету імені Тараса Шевченка
Доктор фізико-математичних наук

ВСТУП

Статистичні опитування розробників сучасного програмного забезпечення [1,2] показали, що мікросервісна архітектура вважається найпрактичним підходом до керування великомасштабними додатками.

Розбиття структури застосунку на велику кількість функціональних частин не тільки дозволяє розділити різні домени сутностей і ізолювати окремі частини бізнес логіки, а й підвищує автономію команд розробників у R&D, щоб надає змогу працювати в окремих депозитаріях коду паралельно. Саме тому сучасні високонавантажені системи часто побудовані з використанням мікросервісної архітектури.

Надійність високонавантажених систем – ключова характеристика, що забезпечує їм здатність функціонувати навіть в умовах динамічних навантажень, локальних збоїв

окремих компонентів і несподіваних втрат доступності.

Відмовостійкість – критично важливий критерій надійності, що забезпечує здатність системи коректно функціонування навіть у разі виходу з ладу окремих компонентів або втрати доступу до певних обчислювальних вузлів [3].

Мікросервісна архітектура (рис. 1) надає переваги при необхідності ізоляції збоїв і підтримці безперервної роботи у випадку часткових відмов.

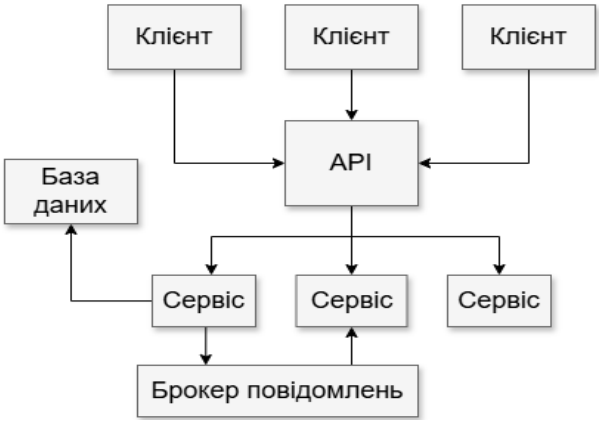

Рис.1. Приклад мікросервісної архітектури високонавантаженої системи [4]

До того ж розподіл функціональності між незалежними сервісами і децентралізація створюють передумови для досягнення оптимальної продуктивності і адаптивності системи. Це, дозволяє значно покращити стабільність роботи застосунків, що функціонують у середовищах з високими вимогами до безперервності і затримок в умовах змінних навантажень [4].

Проте, велика кількість незалежних мікросервісів ускладнює контроль за їхньою взаємодією, оскільки відмова окремих сервісів може бути не одразу виявлена без відповідного спостереження за системним станом. Крім того, розподілений характер мікросервісних додатків може створювати такі системні аномалії, як каскадні відмови [5], що створює нові виклики у забезпеченні надійності, а саме висуває на перший план задачу ефективної оркестрації і моніторингу компонентів. Незалежність мікросервісів робить компоненти системи значною мірою залежними від комунікації, що посилює властивість емерджентності системи і може призводити до значних непередбачуваних негативних наслідків [6]

У відповідь на низку зазначених ризиків, що пов’язані з доступністю застосунку, для зменшення наслідків збоїв в мережі, системних відмов, різких змін навантаження (інтенсивності використання функціоналу користувачами) і порушень безпеки часто застосовують шаблони відмовостійкості.

Існує багато програмних бібліотек і фреймворків [7, 8], що пропонують для деяких шаблонів різні імплементації (реалізації), які розробники програмного забезпечення можуть використовувати «з коробки», змінюючи значення змінних у конфігурації, що надається бібліотекою.

Також існує низка різних шаблонів відмовостійкості, що класифікуються за різними категоріями [7–9]. Проте це дослідження сфокусоване на покращенні тих шаблонів, що безпосередньо впливають на синхронну комунікацію між сервісами.

Серед шаблонів, які опинилися в полі цього дослідження [7, 9]: Timeout; Circuit Breaker; Retries; Fallbacks; Bulkheads; Rate Limiters. В роботі [7] на основі аналізу цих шаблонів для подальшого вдосконалення було виокремлено Circuit Breaker (CB). Також у [7] було запропоновано нову реалізацію цього методу – предиктивну модель патерну Circuit Breaker, і виконано порівняння запропонованої моделі з загальновідомим, реалізованим на базі [8], Circuit Breaker.

У цій реалізації шаблону, для оцінки поточної надійності зв’язку між сервісами використано такі метрики на основі даних про нещодавні виклики: Failure rate (рівень відмов); Slow call rate (низька швидкість запитів); Success call rate (рівень успішних запитів); Time in Open State (час у відкритому стані).

Запропонована в [7] предиктивна модель патерну Circuit Breaker спрямована на оптимізацію синхронної комунікації двох мікросервісів шляхом прогнозування стану їх взаємодії і зменшення затримки переходу між станами Circuit Breaker, що покращує статистику успішності запитів. Однак, цю модель Circuit Breaker можна ще

вдосконалити в напрямку покращення її адаптивності до змін навантаження.

ПІДВИЩЕННЯ ВІДМОВОСТІЙКОСТІ МІКРОСЕРВІСІВ

Передумови вдосконалення методу СВ

Динамічне навантаження на мікросервіс високонавантаженої системи – це типове явище, що залежить від контексту, в якому працює система.

Саме тому більшість сучасних реалізацій Circuit Breaker працюють на основі ковзного вікна даних, що тримає контекст певного розміру за заданим часом або кількістю викликів в залежності від його типу [8, 10]:

- Count-based sliding window (ковзне вікно на основі кількості запитів);
- Time-based sliding window (ковзне вікно на основі часових проміжків);

Кожен тип вікна тримає контекст за певною кількістю останніх викликів, але всі ці реалізацій надають статичний розмір вікна.

Контекст оновлюється на базі даних статистичних останніх запитів і показників продуктивності, що використовуються для оцінки поточної надійності зв'язку між сервісами.

Деякі існуючі фреймворки і бібліотеки пропонують механізм вилучення «поганих екземплярів», але також без автоматичного налаштування розміру вікна. В цій роботі для вдосконалення предиктивної моделі патерну СВ [7] запропоновано динамічне регулювання розміру ковзного вікна на основі поточного навантаження на сервіс. Таке налаштування надає змогу адаптувати контекст до природних коливань трафіку.

Поточним називається навантаження, що спостерігається в поточний інтервал часу.

Для реалізації цієї ідеї, до предиктивної моделі патерну СВ, додано контролер, який змінює цільовий розмір ковзного вікна відповідно до поточних показників інтенсивності викликів функціоналу сервісу.

Це, своєю чергою передбачає:

- розробку алгоритму для адаптивного налаштування розміру вікна даних;
- розробку математичної моделі для обчислення розміру ковзного вікна, на основі якого працює СВ [7, 8].

Алгоритм адаптивного налаштування розміру ковзного вікна вікна даних

В основу роботи контролера, покладено алгоритм, що періодично виконує:

1) обчислення поточного навантаження;
2) експоненціальне згладжування обчисленого поточного навантаження;
3) обчислення цільового розміру вікна;
4) обчислення відносної зміни порогу гістерезису;
5) перевірка умови оновлення розміру цільового вікна;
6) оновлення цільового розміру вікна, при умові перевищення його відносної зміни порогу гістерезису.

Початкове значення розміру вікна W_0 налаштовується під час запуску програми.

Адаптація математичної моделі

1. Обчислення поточного навантаження виконується за формулою (1):

$$\hat{\lambda}_t = \frac{n_t}{\Delta}, \qquad (1)$$

де: n_t – кількість запитів за заданий інтервал часу Δ, а $\hat{\lambda}_t$ – поточне навантаження.

2. Експоненціальне згладжування обчисленого поточного навантаження, що виконується для стабілізації короткочасних шумів і отримання згладженої пропускної здатності, виконується за формулою (2):

$$\lambda_t = (1-\gamma)\lambda_{t-1} + \gamma\hat{\lambda}_t, \quad \gamma \in (0,1], \qquad (2)$$

де: λ_t – згладжене навантаження в момент часу t; λ_{t-1} – згладжене навантаження в попередній часовий інтервал; $\hat{\lambda}_t$ – поточне навантаження; γ – коефіцієнт згладжування.

3. Обчислення розміру вікна в момент часу t виконується за формулою (3):

$$W_{target,t} = clip(\alpha\lambda_t, W_{min}, W_{max}), \qquad (3)$$

де; $W_{target,t}$ – цільовий розмір ковзного вікна в момент часу t; α – коефіцієнт

масштабування, що контролює пропорційну зміну розміру ковзного вікна відносно навантаження; λ_t – згладжений розмір навантаження в момент часу t; $clip$ – функція обмеження розміру вікна; W_{min} і W_{max} – нижня і верхня межі дозволеного розміру вікна, що запобігають надмірному стисненню і розширенню.

4. зміни порогу гістерезису виконується за формулою (4):

$$\varepsilon_t = \frac{|W_{target,t} - W_{t-1}|}{W_{t-1}}, \qquad (4)$$

де ε_t – відносна зміна навантаження в момент часу t; W_{t-1} – розмір ковзного вікна з попереднього інтервалу; $W_{target,t}$ – розмір ковзного вікна (до переналаштування).

5. Розмір вікна оновлюється згідно з (5):

$$W_t = W_{target,t} \ if \ \varepsilon_t > \varepsilon; \\ otherwise \ W_t = W_{t-1}, \qquad (5)$$

де W_t – розмір вікна, що застосовується в момент часу t; W_{t-1} – розмір ковзного вікна з попереднього інтервалу; $W_{target,t}$ – нове значення розміру ковзного вікна до переналаштування; ε – поріг гістерезису, який визначає мінімальну відносну зміну, що необхідна для оновлення поточного W_{t-1} значення на обчислене W_t.

Умова оновлення розміру ковзного вікна потрібна для уникнення невиправдано частих коливань, які можуть бути викликані мінорними змінами в трафіку.

6. Оновлення розміру ковзного вікна запускається в окремому від основних процесів роботи Circuit Breake потоці, або ініціюється фоновим планувальником через спеціальний таймер.

Період оновлення розміру ковзного вікна визначається відповідно до потреб системи.

Приклад роботи алгоритму

На рис. 2 показано динаміку HTTP запитів, що відповідає реальній статистиці добових коливань трафіку серверів Cloudflare у Європі протягом тижня у вересні 2025 року [11].

З рис.2 видно як інтенсивність трафіку перевищує 90% вдень (рис. 2, а), а вночі зменшується до 30% (рис. 2, б).

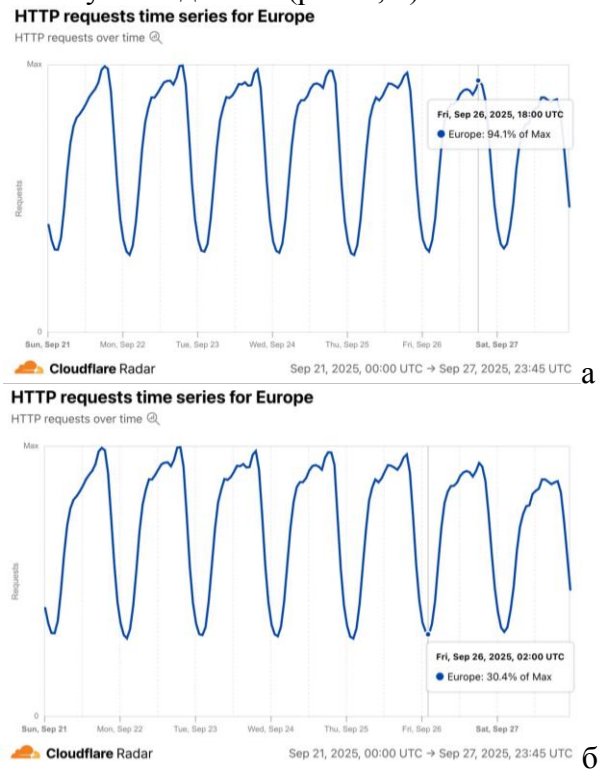

Рис. 2. Статистика добових коливань HTTP запитів до серверів Cloudflare у Європі [11]

На рис. 3 показано приклад динаміки розміру адаптивного і статичного ковзних вікон в умовах періодичних навантажень.

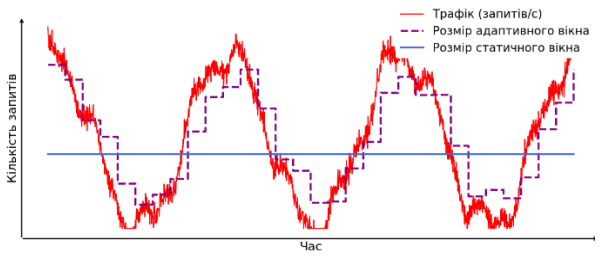

Рис. 3. Приклад динаміки розміру адаптивного і статичного ковзних вікон

З рис. 3 видно, що запропонований алгоритм адаптує розмір ковзного вікна відповідно до поточного навантаження, а саме:

- Якщо навантаження на мікросервіс зростає, то розмір вікна збільшується;
- Якщо навантаження спадає, то розмір вікна зменшується.

Залежність динаміки розміру ковзного вікна від трафіку даних надає ширший контекст для аналізу стану комунікації між

сервісами у порівнянні з статистичними вікнами, що нечутливі до цих змін.

Таким чином «пробна перевірка» роботи алгоритму показала, що алгоритм відповідає очікуванням.

Проте, в цьому дослідженні залишились невирішеними задачі:

- експериментальної оцінки здатності предиктивної моделі патерну CB адаптуватися до реальних змін робочого навантаження;
- порівняння ефективності роботи предиктивної моделі патерну CB в різних сценаріях роботи високонавантажених систем, в тому числі використовуючи поєднання цієї реалізації шаблону з іншими патернами відмовостійкості.

Для порівняння результатів цих експериментів пропонується використати метрики рейтингу комунікації сервісів: Slow call rate, щ показує низьку швидкість запитів; Success call rate, яка відображає рівень успішних викликів; Time in the Open state, яка показує час відкритого стану Circuit Breaker; Permitted call rate, що показує кількість дозволених до виконання Circuit Breaker спроб запитів з тих що записані в ковзному вікні; Consecutive failure streak, яка відображає поточну серія послідовних невдач.

Таким чином, перспективою подальших розробок є експериментальні дослідження, в яких запропонований алгоритм буде використано в різних прикладних сценаріях.

ВИСНОВКИ

1. Основним результатом дослідження є алгоритм налаштування розміру цільового ковзного вікна, в основу роботи якого покладено математичну модель, що розроблена для застосування предиктивною моделлю патерну Circuit Breaker.

2. Запропонований алгоритм адаптує розмір цільового ковзного вікна відповідно до поточного навантаження на мікросервіс системи, використовуючи експоненційне згладжування і поріг гістерезису для уникнення занадто частих і різких змін розміру контексту.

3. Для оцінки ефективності алгоритму в складніших конфігураціях мікросервісів заплановано проведення експериментальні досліджень, в яких його буде використано в різних прикладних сценаріях.

REFERENCES

1. **Stack Overflow.** (2024). Stack Overflow Developer Survey 2023. https://survey.stackoverflow.co/2023/
2. **JetBrains.** (2024). General Development Trends - The State of Developer Ecosystem in 2023 infographic. https://www.jetbrains.com/lp/devecosystem-2023/development/
3. **Siunduh M. E. S., Otieno, M. V. M., Mbugua P. S.** (2025). Fault-Tolerant Software architecture: A comprehensive analysis of design patterns, implementation strategies and performance evaluation. Zenodo (CERN European Organization for Nuclear Research). https://doi.org/10.5281/zenodo.15868925
4. **Bugrov A. A., Terenchuk S. A.** (2023). Principi i metodi avtomatichnogo masshtabuvannja visokonavantazhenih sistem. *Shljahi pidvishhennja efektivnosti budivnictva v umovah formuvannja rinkovih vidnosin*, 52(3), 217-226. https://doi.org/10.32347/2707-501x.2023.52(3).217-226
5. **Soldani J., Forti S., Roveroni L., Brogi A.** (2024). Explaining microservices' cascading failures from their logs. Software Practice and Experience, 55(5), 809–828. https://doi.org/10.1002/spe.3400
6. **Lewis J., Fowler M**. (2014). Microservices. https://martinfowler.com/articles/microservices.html
7. **Hlybovets A., Paprotskyi I.** (2024). Increasing the fault tolerance in microservice architecture. *Cybernetics and Systems Analysis*, 60(3), 480–488. https://doi.org/10.1007/s10559-024-00689-0
8. **Resilience4j.** (2025). Introduction. https://resilience4j.readme.io/docs/getting-started
9. **Nygard M. T.** (2018). Release it!: Design and Deploy Production-Ready Software (2nd ed.). *The Pragmatic Programmers LLC*.
10. **Resilience4j.** (2025). Circuit Breaker. https://resilience4j.readme.io/docs/circuitbreaker
11. **Cloudflare** (2025). Data Explorer | CloudFlare Radar. (2025, September 21). https://radar.cloudflare.com/explorer?dataSet=http&dt=2025-09-21_2025-09-27&loc=europe.

Improving the Fault Tolerance of Microservices in Highly Loaded Systems Based on Load Monitoring

Ihor Paprotskyi, Anatolii Pashko

Abstract. The article investigates the problem of configuring communication between servers in high-load systems that operate under periodic conditions. The primary attention is on fault-tolerance patterns that directly affect synchronous communication between services. The study focuses on the Circuit Breaker method, which formalizes the analogy to a real circuit breaker for fault control and isolation in distributed systems.

The goal of the work is to increase the fault tolerance of a microservice by dynamically adjusting the sliding data collection window size, enabling the context to adapt to current traffic fluctuations.

To achieve this goal, a mathematical model for calculating the target sliding window size has been developed based on the load analysis over the observed time interval and incorporated into Circuit Breaker.

The algorithm periodically calculates and adapts the size of the sliding data collection window to traffic fluctuations in the system, based on current load calculations and exponential smoothing, as described in the proposed model.

The practical significance of the results obtained is seen in the possibility of implementing new algorithms to improve the communication rating of microservices in highly loaded systems, which will allow for increasing their fault tolerance and ensuring system stability under different load conditions. The scientific novelty lies in the further development of the methodology for increasing the fault tolerance of microservices in highly loaded systems, based on the determination of their interaction state using statistical data from previous requests and their outcomes.

Keywords: adaptivity, Circuit Breaker, fault tolerance pattern, synchronous communication, sliding window, traffic.