МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Кваліфікаційна наукова
праця на правах рукопису

БУГРОВ АНАТОЛІЙ АНАТОЛІЙОВИЧ

УДК 004.2; 004.6; 004.9

ДИСЕРТАЦІЯ

МОДЕЛІ І МЕТОДИ ВДОСКОНАЛЕННЯ ВИСОКОНАВАНТАЖЕНИХ РОЗПОДІЛЕНИХ СИСТЕМ

126 – Інформаційні системи та технології
12 – Інформаційні технології

Подається на здобуття наукового ступеня **доктора філософії**

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело
_____ А. А. Бугров

Науковий керівник **Єременко Богдан Михайлович,**
кандидат технічних наук, доцент кафедри інформаційних технологій проектування та прикладної математики Київського національного університету будівництва і архітектури

Київ-2025

АНОТАЦІЯ

Бугров А.А. **Моделі і методи вдосконалення високонавантажених розподілених систем.** – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 126 «Інформаційні системи та технології». – Київський національний університет будівництва і архітектури. – Київ, 2025.

Наукова новизна одержаних результатів. Наукова новизна дисертаційної роботи полягає у тому, що:

вперше розроблено:

– математичну модель динамічного масштабування ресурсів, що поєднує експоненційне згладжування для прогнозу із нечітким агрегуванням лінгвістичних оцінок та дозволяє уникнути фіксованих порогів, скорочуючи час реакції високонавантажених розподілених систем;

– адаптивний нечіткий регресивний метод (АНРМ) застосування зазначеної моделі, що забезпечує безперервне формування керуючих впливів шляхом нечіткої регресивної оцінки параметрів масштабування ресурсів;

– метод коригування моменту масштабування, який, аналізуючи вихідні величини АНРМ, визначає найкращий час зміни конфігурації ресурсів і забезпечує баланс між продуктивністю та експлуатаційними витратами;

удосконалено:

– метод оцінювання ефективності масштабування ресурсів, доповнений багатокритеріальним аналізом технічних і економічних показників, що забезпечує комплексну оцінку результативності керування ресурсами;

набули подальшого розвитку:

– концепція комбінованих стратегій управління ресурсами, у межах якої прогнозні процедури експоненційного згладжування інтегровано з нечіткими реактивними механізмами, що підвищує гнучкість і своєчасність адаптації системи;

– методи аналізу впливу динаміки навантаження на результативність масштабування, що уточнюють критерії ініціювання процесу залежно від швидкості та амплітуди зміни навантаження.

Основний зміст дисертаційної роботи

Дисертаційна робота присвячена проблемам адаптивного управління ресурсами високонавантажених розподілених систем, що виникають в умовах динамічних змін навантаження, необхідності балансування між продуктивністю, масштабованістю та відмовостійкістю, а також економічної ефективності.

За результатами дослідження розроблено:

– математичну модель динамічного масштабування ресурсів, що поєднує експоненційне згладжування для прогнозу із нечітким агрегуванням лінгвістичних оцінок та дозволяє уникнути фіксованих порогів, скорочуючи час реакції високонавантажених розподілених систем;

– адаптивний нечіткий регресивний метод (АНРМ) застосування зазначеної моделі, що забезпечує безперервне формування керуючих впливів шляхом нечіткої регресивної оцінки параметрів масштабування ресурсів;

– метод коригування моменту масштабування, який, аналізуючи вихідні величини АНРМ, визначає найкращий час зміни конфігурації ресурсів і забезпечує баланс між продуктивністю та експлуатаційними витратами.

У першому розділі на основі аналізу сучасних високонавантажених розподілених систем встановлено, що їх ефективність і стійкість залежать від архітектурних рішень, механізмів керування ресурсами і вибору технологічного стеку; охарактеризовано широке коло застосувань, зокрема хмарні обчислення, обробку потокових даних і мікросервісні архітектури; з’ясовано, що головними проблемами є баланс між продуктивністю, масштабованістю та відмовостійкістю, що вимагає оптимізації як на рівні інфраструктури, так і програмного забезпечення; досліджено методи масштабування й управління ресурсами, серед яких виділено вертикальне та горизонтальне масштабування: перше дає змогу підвищити продуктивність окремих вузлів, однак має фізичні обмеження, тоді як друге потребує складніших механізмів балансування навантаження, зате забезпечує вищу

відмовостійкість; розглянуто критерії надійності та метрики ефективності, що охоплюють пропускну здатність, час відгуку, хвостові затримки, коефіцієнт відмов і середній час відновлення після збою, а також економічну ефективність, яка вимірюється рівнем використання ресурсів та вартістю обробки запитів; окреслено, що ефективне управління ресурсами й методи адаптивного масштабування дають можливість знизити витрати без втрати продуктивності, а масштабованість системи безпосередньо впливає на її спроможність адаптуватися до змін навантаження.

У другому розділі проаналізовано існуючі методи автоматичного масштабування високонавантажених систем і визначено необхідність адаптивного управління ресурсами в умовах аномальних і непередбачуваних навантажень; розглянуто вплив динамічних змін у запитах на ефективність масштабування та виявлено, що жорсткі порогові значення можуть призводити до неефективного розподілу ресурсів, що підтверджує доцільність використання більш гнучких стратегій масштабування; досліджено реактивні методи автоматичного масштабування, серед яких метод порогових правил, автоматичне масштабування на основі теорії черг та адаптивні реактивні методи; показано, що ці методи є ефективними у випадках різких змін навантаження, однак вони мають певні обмеження, пов’язані з запізненням у реакції на зміну системних параметрів; обґрунтовано необхідність комбінування реактивних методів із проактивними підходами для підвищення ефективності управління ресурсами; розглянуто можливості застосування проактивних методів автоматичного масштабування, зокрема машинного навчання, розширеного навчання з підкріпленням, нечіткої логіки та аналізу часових рядів; показано, що використання проактивних методів дозволяє передбачати майбутні навантаження та ініціювати масштабування завчасно, що зменшує ризик перевантаження системи; обґрунтовано необхідність розробки комбінованих стратегій, які б дозволяли поєднувати прогнозні підходи з механізмами негайного реагування на зміну навантаження.

У третьому розділі запропоновано адаптивну нечітку регресивну модель динамічного масштабування ресурсів; на основі цієї моделі розроблено адаптивний нечіткий регресивний метод (AHPM), який забезпечує гнучке керування

обчислювальними ресурсами високонавантажених розподілених систем; обґрунтовано доцільність застосування нечіткої логіки для оцінювання поточного стану навантаження, що дає змогу уникнути жорстких порогових значень і здійснювати плавне масштабування; у межах АНРМ розроблено систему нечіткого виведення й сформовано матрицю ваг термів; проведено дослідження впливу характеристик хвилі навантаження на ефективність масштабування і показано, що невдалий вибір моменту зміни конфігурації може спричинити перевитрати або зниження продуктивності; розроблено метод коригування моменту масштабування, який аналізує вихідні величини АНРМ і визначає найкращий час зміни конфігурації, запобігаючи частому перемиканню станів системи; описано принцип використання розрахованого оптимального часу згортання ресурсів; показано, що застосування запропонованого методу, який інтегрує експоненційне згладжування з подвійною корекцією й нечітку регресивну оцінку, усуває надмірну реактивність класичних методів та забезпечує стабільну динаміку адаптації ресурсів до змін навантаження.

У четвертому розділі проаналізовано результати експериментальних досліджень, які підтвердили ефективність запропонованого адаптивного нечіткого регресивного методу динамічного масштабування ресурсів, що поєднує експоненційне згладжування для прогнозу навантаження з нечітким агрегуванням лінгвістичних оцінок; метод дає змогу своєчасно виділяти ресурси, уникати фіксованих порогів і забезпечувати стабільну роботу високонавантажених розподілених систем за оптимального використання обчислювальних потужностей; на прикладі геоінформаційного модуля системи відновлення пошкоджених будівель продемонстровано переваги проактивного масштабування, завдяки якому короткочасні пікові періоди нівелювалися завчасним збільшенням ресурсів і подальшим поверненням до економнішого режиму; для інтелектуальної системи керування транспортним трафіком великого міста показано, що повторюваність годин «пік» полегшує прогнозування, а метод адаптивно коригує обсяг ресурсів під поточний рівень навантаження; на високонавантаженій платформі аналізу даних із соціальних мереж підтверджено здатність методу плавно масштабувати ресурси при поступовому довготривалому зростанні користувацької активності; запропонований

метод може бути інтегрований у практичні системи з високою варіативністю навантаження, забезпечуючи гнучкий і своєчасний розподіл обчислювальних потужностей.

Ключові слова: балансування навантаження, високе навантаження, інформаційні системи, керування ресурсами, нечітка логіка, проактивне масштабування, прогнозування, регресивне моделювання, розподілені системи, часові ряди.

ABSTRACT

Buhrov A. Models and Methods for Improving High-Load Distributed Systems. – Qualified scientific work on the rights of the manuscript.

Dissertation for obtaining the scientific degree of Doctor of Philosophy in specialty 126 "Information systems and technologies". - Kyiv National University of Construction and Architecture. - Kyiv, 2025.

The scientific novelty of the obtained results. The scientific novelty of the dissertation is that:

first developed:

– a mathematical model of dynamic resource scaling that combines exponential smoothing for load forecasting with fuzzy aggregation of linguistic assessments, thereby eliminating fixed thresholds and reducing the reaction time of high-load distributed systems;

– an adaptive fuzzy regression method (AFRM) for applying the proposed model, which ensures continuous generation of control actions through fuzzy-regression evaluation of resource-scaling parameters;

– a method for adjusting the scaling moment, which analyses the output values of AFRM to determine the optimal time for reconfiguring resources, maintaining a balance between performance and operational costs;

improved:

– a method for evaluating the effectiveness of resource scaling, expanded with a multi-criteria analysis of technical and economic indicators, providing a comprehensive assessment of resource-management performance;

acquired further development:

– the concept of combined resource-management strategies, in which forecasting procedures based on exponential smoothing are integrated with fuzzy reactive mechanisms, enhancing the flexibility and timeliness of system adaptation;

– methods for analysing the influence of load dynamics on scaling efficiency, which refine the criteria for initiating scaling depending on the rate and amplitude of load variations.

The main content of the dissertation

The dissertation addresses problems of adaptive resource management in high-load distributed systems arising under significant dynamic load changes, the necessity of balancing performance, scalability, and fault tolerance, as well as economic efficiency.

Based on the results of the study, the following was developed:

– a mathematical model of dynamic resource scaling that combines exponential smoothing for load forecasting with fuzzy aggregation of linguistic assessments, thereby eliminating fixed thresholds and reducing the reaction time of high-load distributed systems;

– an Adaptive Fuzzy Regression Method (AFRM) for applying the proposed model, which ensures continuous generation of control actions through fuzzy-regression evaluation of resource-scaling parameters;

– a method for adjusting the scaling moment, which analyses the output values of AFRM to determine the optimal time for reconfiguring resources, maintaining a balance between performance and operational costs.

In the first chapter, based on an analysis of modern high-load distributed systems, it has been established that their efficiency and resilience depend on architectural decisions, resource management mechanisms, and the choice of technology stack; a wide range of applications has been characterized, including cloud computing, stream data processing, and microservice architectures. It has been found that the main challenges are balancing performance, scalability, and fault tolerance, requiring optimization at both the infrastructure and software levels. Methods of scaling and resource management have been investigated, among which vertical and horizontal scaling have been highlighted: the former allows increasing the performance of individual nodes but has physical limitations, while the latter requires more complex load-balancing mechanisms but provides higher fault tolerance. Reliability criteria and performance metrics have been examined, covering throughput, response time, tail latency, failure rate, and mean time to recovery, as well as

cost-effectiveness measured by the level of resource utilization and the cost of processing requests. It has been outlined that effective resource management and adaptive scaling algorithms can reduce costs without sacrificing performance, and that system scalability directly affects its ability to adapt to changing loads.

The second chapter analyzed existing approaches to automatic scaling of high-load systems and identified the necessity of adaptive resource management under abnormal and unpredictable loads. The impact of dynamic changes in requests on scaling efficiency has been discussed, revealing that rigid thresholds can lead to inefficient resource allocation, confirming the advisability of more flexible scaling strategies. Reactive methods of automatic scaling have been investigated, including the threshold rule method, automatic scaling based on queue theory, and adaptive reactive algorithms. It has been shown that these methods are effective in cases of sharp load changes but have certain limitations related to delayed responses to changes in system parameters. The necessity of combining reactive methods with proactive approaches to enhance resource management efficiency has been substantiated. The potential of proactive automatic scaling methods has been considered, particularly machine learning, advanced reinforcement learning, fuzzy logic, and time series analysis. It has been demonstrated that the use of proactive methods makes it possible to predict future loads and initiate scaling in advance, thereby reducing the risk of system overload. The necessity of developing combined strategies that allow integrating forecasting approaches with immediate response mechanisms to load changes has been justified.

In the third chapter, an adaptive fuzzy regression model for dynamic resource scaling is proposed; on the basis of this model an adaptive fuzzy regression method (AFRM) is developed, which provides flexible control of computing resources in high-load distributed systems; the expediency of using fuzzy logic for assessing the current load state is substantiated, enabling the avoidance of hard threshold values and the implementation of smooth scaling; within AFRM a fuzzy inference system is devised and a weight matrix of terms is formed; a study of the influence of load-wave characteristics on scaling efficiency is carried out, demonstrating that an improper choice of scaling moment may cause cost

overruns or performance degradation; a method for adjusting the scaling moment is developed, which analyses AFRM outputs and determines the optimal time to change the resource configuration, preventing frequent state switching in the system; the principle of using the calculated optimal resource-contraction time is described; it is shown that the proposed method, integrating double-corrected exponential smoothing with fuzzy regression evaluation, eliminates the excessive reactivity of classical autoscalers and ensures stable adaptation dynamics of resources to load fluctuations.

The fourth chapter analyzed experimental research results, confirming the effectiveness of the proposed adaptive fuzzy-regressive method of dynamic resource scaling, which combines exponential smoothing for load prediction with fuzzy aggregation of linguistic assessments; the method enables timely resource allocation, avoids fixed thresholds and ensures stable operation of high-load distributed systems while optimally utilising computing capacity; using the geo-information module of a post-disaster building-recovery system the advantages of proactive scaling are demonstrated, whereby short-term peak periods were mitigated by anticipatory resource increase followed by a return to a cost-efficient regime; for an intelligent urban traffic-management system it is shown that the regular repetition of rush hours facilitates prediction, and the method adaptively adjusts resource volume to the current load level; on a high-load distributed social-media-analytics platform the ability of the method to smoothly scale resources during a gradual long-term growth of user activity is confirmed; the proposed approach can be integrated into real-world systems with highly variable load, providing flexible and timely distribution of computing power.

Keywords: regression modeling, fuzzy logic, time series, resource management, forecasting, proactive scaling, information systems, distributed systems, high load, high performance, load balancing.

СПИСОК ОПУБЛІКОВАНИХ НАУКОВИХ ПРАЦЬ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Статті у наукових періодичних виданнях інших держав та у виданнях України, які включено до міжнародних наукометричних баз

1. Yeremenko B., Mazurenko R., Stetsyk O., **Buhrov A.** (2023) Intelligent Management of Traffic Flows in Large Cities. Lecture Notes in Intelligent Transportation and Infrastructure, Part F1379, 33-42. DOI: https://doi.org/10.1007/978-3-031-25863-3_4 (Scopus, ISSN: 2523-3440)
2. **Бугров А.**, Теренчук С. (2023). Принципи і методи автоматичного масштабування високонавантажених систем. Шляхи підвищення ефективності будівництва. No52(3). С. 217-226. DOI: https://doi.org/10.32347/2707-501x.2023.52(3).217-226 (Фахове видання категорії «Б»)
3. **Бугров А.,** Волох Б., Босенко І., Теренчук С. (2024). Система підтримки процесу відновлення об’єктів нерухомості: обробка і збереження даних. Управління розвитком складних систем, (60), 136–145. DOI: https://doi.org/10.32347/2412-9933.2024.60.136-145 (Фахове видання категорії «Б»)
4. **Бугров А.,** Оптимізація геоінформаційного сервісу в системі підтримки процесу відновлення об’єктів нерухомості. Управління розвитком складних систем, (61), 187–192. DOI: https://doi.org/10.32347/2412-9933.2025.61.187-192 (Фахове видання категорії «Б»)

Наукові праці, що представлені як тези доповіді у міжнародних науково-технічних конференціях

5. Terenchuk S., Pasko R., **Buhrov A.,** Ploskyi V., Panko O., Zapryvoda V. (2022). Computerization of the process of reconstruction of damaged or destroyed real estate. 2022 IEEE 3rd KhPI Week on Advanced Technology, KhPI Week 2022 – Conference Proceedings. DOI: https://doi.org/10.1109/KhPIWeek57572.2022.9916470 (Scopus)
6. **Buhrov A.,** Pasko R., Yeremenko B. Computerization of the Process of Assessing the Technical Condition of Buildings and Infrastructure. ACeSYRI – International

Workshop on Modern Experience for PhD students and Young Researchers, November 14-18, 2022, p. 17, book of abstract, Zilina, Slovakia. https://ki.fri.uniza.sk/ACeSYRI2022/Abstracts.pdf

7. Terenchuk S., **Buhrov A.,** Pasko R., Yaschenko A., Bosenko I., Volokh B. (2023). Ontology Formation of Support System for Restoration of Buildings, Property and Infrastructure Objects. 2023 IEEE 4th KhPI Week on Advanced Technology, KhPI Week 2023 – Conference Proceedings. DOI: https://doi.org/10.1109/KhPIWeek61412.2023.10313006 (Scopus)

ЗМІСТ

ВСТУП

Актуальність теми. Дисертаційна робота присвячена проблемам адаптивного керування ресурсами у високонавантажених розподілених системах, які стикаються з викликами динамічних і непередбачуваних змін навантаження, балансування продуктивності, масштабованості та економічної ефективності.

Підвищення ефективності та стабільності роботи таких систем є критично важливим завданням у багатьох галузях, включаючи хмарні сервіси, мікросервісні архітектури та системи обробки потокових даних, де зростаючі обсяги інформації вимагають швидкого і точного розподілу ресурсів і потребують індивідуального підходу до масштабування і оптимізації, який забезпечить високу продуктивність, адаптивність і економічність системи.

Таким чином розробка адаптивних засобів керування ресурсами спеціальних високонавантажених ситем, робота яких базується на гібридних методах автоматичного масштабування, залишається актуальною.

Зв’язок роботи з науковими програмами, планами, темами. Наукова робота відповідає напрямку, визначеному у Законі України «Про Національну програму інформатизації» (2023 р.), відповідає цілям Національної стратегії розвитку інформаційних технологій до 2030 року і може бути інтегрована у програму «Цифрова Україна». Дисертація відповідає паспорту спеціальності 126 – Інформаційні системи та технології.

Дисертація відповідає тематичному спрямуванню наукових розробок в рамках науково-дослідної роботи кафедри інформаційних технологій проектування та прикладної математики факультету автоматизації і інформаційних технологій Київського національного університету будівництва і архітектури та пов’язана із планами науково-дослідних робіт кафедри. Результати дослідження впроваджено в навчальний процес в межах Київського національного університету будівництва і архітектури.

Зокрема, в навчальні програми дисциплін:

– «Інформаційні технології представлення обробки та розпізнавання зображень» для магістрів спеціальності «Інформаційні системи та технології» (ІСТм-23) кафедри інформаційних технологій проектування та прикладної математики.

– «Прикладна теорія графів» для магістрів спеціальності «Автоматизація та комп'ютерно-інтегровані технології» (АКІТм-24) кафедри автоматизації технологічних процесів.

Дисертація містить наукові положення, нові науково обґрунтовані теоретичні результати проведених досліджень, які мають істотне значення для галузі знань 12 – Інформаційні технології.

Мета дослідження полягає в розробці адаптивної нечіткої регресивної моделі та відповідного методу динамічного масштабування ресурсів для високонавантажених розподілених систем, що здатні підтримувати стабільну продуктивність, зменшити витрати обчислювальних ресурсів і оперативно адаптуватися до змін навантаження.

Для досягнення вказаної мети необхідно вирішити такі завдання:

1. Проаналізувати архітектури сучасних високонавантажених розподілених систем і визначити ключові фактори, що впливають на їх ефективність — продуктивність, відмовостійкість та економічність.

2. Оцінити переваги й обмеження наявних методів автоматичного масштабування ресурсів у середовищах з динамічними змінами навантаження.

3. Розробити адаптивну нечітку регресивну модель динамічного масштабування та відповідний метод, які усувають потребу у фіксованих порогах і забезпечують гнучке керування.

4. Сформувати метод прогнозування навантаження на основі експоненційного згладжування та інтегрувати його з розробленою моделлю для оптимального розподілу ресурсів.

5. Розробити метод коригування методу масштабування, що визначає найкращий момент зміни конфігурації ресурсів на підставі вихідних величин методу.

6. Провести експериментальну оцінку ефективності розроблених методів за критеріями продуктивності, відмовостійкості та економічної доцільності.

Об’єкт дослідження – високонавантажені розподілені системи, які працюють із великими обсягами даних в умовах змінного навантаження.

Предмет дослідження – моделі та методи адаптивного масштабування обчислювальних ресурсів у високонавантажених розподілених системах.

Методи дослідження, що використовувалися під час роботи:

- критичний аналіз існуючої наукової літератури при виборі методів автоматичного масштабування для формування адаптивного нечіткого регресивного методу;
- синтез реактивного та проактивного методів для створення гібридного методу автоматичного масштабування;
- формалізація технічних параметрів та процесів у вигляді змінних і математичних залежностей;
- імітаційне моделювання для оцінки якості роботи запропонованих методів.

Наукова новизна одержаних результатів. Наукова новизна дисертаційної роботи полягає у тому, що:

вперше розроблено:

– математичну модель динамічного масштабування ресурсів, що поєднує експоненційне згладжування для прогнозу із нечітким агрегуванням лінгвістичних оцінок та дозволяє уникнути фіксованих порогів, скорочуючи час реакції високонавантажених розподілених систем;

– адаптивний нечіткий регресивний метод (АНРМ) застосування зазначеної моделі, що забезпечує безперервне формування керуючих впливів шляхом нечіткої регресивної оцінки параметрів масштабування ресурсів;

– метод коригування моменту масштабування, який, аналізуючи вихідні величини АНРМ, визначає найкращий час зміни конфігурації ресурсів і забезпечує баланс між продуктивністю та експлуатаційними витратами;

удосконалено:

– метод оцінювання ефективності масштабування ресурсів, доповнений багатокритеріальним аналізом технічних і економічних показників, що забезпечує комплексну оцінку результативності керування ресурсами;

набули подальшого розвитку:

– концепція комбінованих стратегій управління ресурсами, у межах якої прогнозні процедури експоненційного згладжування інтегровано з нечіткими реактивними механізмами, що підвищує гнучкість і своєчасність адаптації системи;

– методи аналізу впливу динаміки навантаження на результативність масштабування, що уточнюють критерії ініціювання процесу залежно від швидкості та амплітуди зміни навантаження.

Практичне значення отриманих результатів – полягає в можливості застосування запропонованих методів для підвищення ефективності управління ресурсами у високонавантажених системах різного призначення. Зокрема, в геоінформаційному сервісі системи підтримки процесу відновлення об’єктів нерухомості. Це дає змогу зменшити витрати на серверну інфраструктуру, підвищити ефективність обробки даних та забезпечити стабільну роботу системи за різних умов навантаження. У випадку інтелектуальної системи керування транспортним трафіком великого міста доведено, що повторюваність годин «пік» сприяє точнішому прогнозуванню, а метод забезпечує адаптивне коригування обсягу ресурсів відповідно до поточного рівня навантаження. На прикладі високонавантаженої платформи для аналізу даних із соціальних мереж підтверджено здатність методу забезпечувати плавне масштабування ресурсів за умов поступового й тривалого зростання активності користувачів.

Особистий внесок здобувача. Дисертація є самостійною науковою працею, в якій висвітлені власні ідеї та розробки автора, що дозволили досягти поставленої мети. З наукових праць, опублікованих у співавторстві, у дисертації використано лише ті ідеї та положення, які є результатом особистої роботи здобувача.

Апробація результатів дисертації. Основні положення та результати дисертаційної роботи висвітлені та обговоренні на наступних конференціях:

1. 2022 IEEE 3rd KhPI Week on Advanced Technology, KhPI Week 2022 – Conference Proceedings. DOI: https://doi.org/10.1109/KhPIWeek57572.2022.9916470

2. ACeSYRI – International Workshop on Modern Experience for PhD students and Young Researchers, November 14-18, 2022, pp. 40-41, book of abstract, Zilina, Slovakia. URL: https://ki.fri.uniza.sk/ACeSYRI2022/Abstracts.pdf

3. 2023 IEEE 4th KhPI Week on Advanced Technology, KhPI Week 2023 - Conference Proceedings. DOI: https://doi.org/10.1109/KhPIWeek61412.2023.10313006

В повному обсязі дисертація доповідалась на науковому семінарі кафедри інформаційних технологій, кафедри інформаційних технологій проектування та прикладної математики Київського національного університету будівництва і архітектури (м. Київ, 2025 рік).

Структура та обсяг дисертації. Дисертаційна робота складається із вступу, чотирьох розділів, загальних висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 152 сторінок, у тому числі основна частина складає 123 сторінки. Основна частина, крім тексту, включає 11 таблиць, 25 рисунків.

Подяка. Висловлюю глибоку подяку науковому керівнику – кандидату технічних наук, Єременку Богдану Михайловичу за настанови при написанні роботи.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ВИСОКОНАВАНТАЖЕНИХ РОЗПОДІЛЕНИХ СИСТЕМ І ПІДХОДІВ ДО ЇХ ВДОСКОНАЛЕННЯ

У розділі на основі аналізу сучасних високонавантажених розподілених систем встановлено, що їх ефективність і стійкість залежать від архітектурних рішень, механізмів керування ресурсами і вибору технологічного стеку; охарактеризовано широке коло застосувань, зокрема хмарні обчислення, обробку потокових даних і мікросервісні архітектури; з’ясовано, що головними проблемами є баланс між продуктивністю, масштабованістю та відмовостійкістю, що вимагає оптимізації як на рівні інфраструктури, так і програмного забезпечення; досліджено методи масштабування й управління ресурсами, серед яких виділено вертикальне та горизонтальне масштабування: перше дає змогу підвищити продуктивність окремих вузлів, однак має фізичні обмеження, тоді як друге потребує складніших механізмів балансування навантаження, зате забезпечує вищу відмовостійкість; розглянуто критерії надійності та метрики ефективності, що охоплюють пропускну здатність, час відгуку, хвостові затримки, коефіцієнт відмов і середній час відновлення після збою, а також економічну ефективність, яка вимірюється рівнем використання ресурсів та вартістю обробки запитів; окреслено, що ефективне управління ресурсами й методи адаптивного масштабування дають можливість знизити витрати без втрати продуктивності, а масштабованість системи безпосередньо впливає на її спроможність адаптуватися до змін навантаження.

1.1. Високонавантажені розподілені системи

За оцінками глобальної платформи даних і бізнес-аналітики Statista [1], кількість користувачів мережі інтернет останні 10 років характеризується стійким зростанням і на період лютого 2025 року Інтернетом користується приблизно 67.8% населення планети (рис. 1.1).

Згідно зі статистикою Cisco Systems [2], середньосвітовий інтернет-трафік з 2024 по 2027 рік збільшиться в 3,2 рази (середньорічний темп зростання складе 26%).

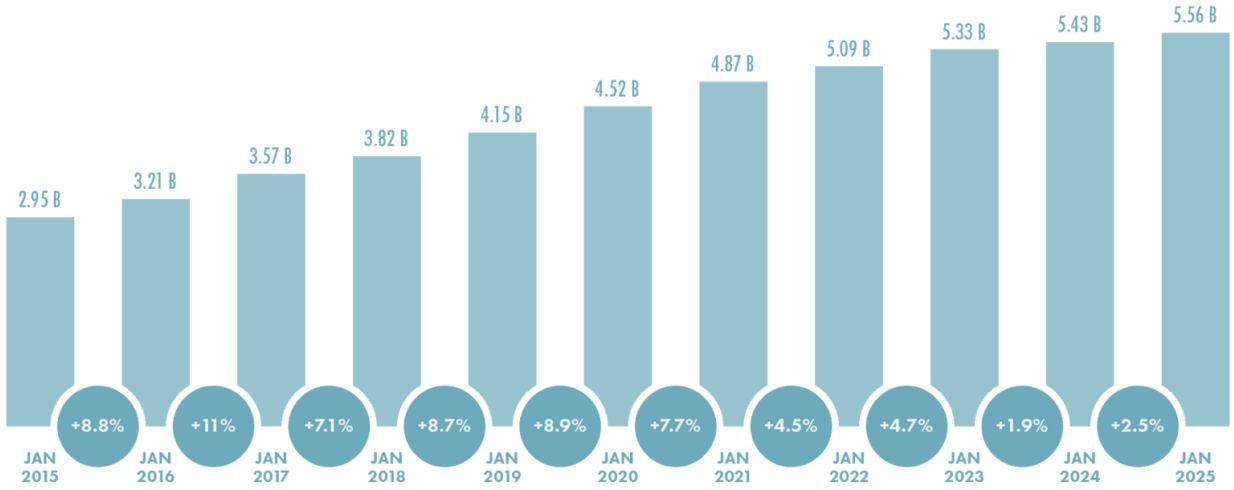

Рисунок 1.1. Динаміка кількості користувачів мережі інтернет за останні 10 років [3]

Наразі відомі високонавантажені системи включають платформи потокового відео (Netflix, YouTube), великі соціальні мережі (Facebook, Twitter) і хмарні сервіси (Amazon Web Services, Microsoft Azure), які демонструють здатність обробляти величезні обсяги даних з високою надійністю та швидкістю.

Найбільшими системами за об’ємом споживання трафіку (рис. 1.2):

1. Google Search Engine, що обробляє понад 3,5 мільярди запитів на день, використовуючи складну архітектуру та розподілені системи.

2. Netflix Streaming Services, що для стрімінгу відео для більше ніж 200 мільйонів користувачів по всьому світу і обробки великих об'ємів даних активно використовує мікросервісну архітектуру, AWS і Apache Kafka.

3. Amazon's E-commerce Platform обробляє мільйони транзакцій на день, використовуючи власні розподілені системи та AWS.

4. Twitter's Social Networking Services використовує такі технології, як Apache Cassandra і Redis, для обробки великої кількості твітів, лайків та ретвітів щосекунди.

5. Facebook's Social Media Platform використовує розподілену інфраструктуру для обслуговування своїх 2,8 мільярдів активних користувачів щомісяця, включаючи використання Apache Hadoop для обробки великих даних.

Рис. 1.2 також показує різноманітність застосувань і промислові сектори, що варіюють від пошукових машин до соціальних мереж, в яких високонавантажені розподілені системи відіграють ключову роль.

APP TOTAL VOLUME 2022

	Application	Total Volume
1	Netflix	13.74%
2	YouTube	10.51%
3	Generic QUIC	5.41%
4	HTTP Media Stream	4.33%
5	Disney+	4.20%
6	Tik Tok	3.55%
7	Facebook	2.83%
8	Xbox Live	2.71%
9	Playstation Downloads	2.70%
10	Amazon Prime	2.67%

а

DOWNSTREAM TRAFFIC

	Application	Total Volume
1	Netflix	14.93%
2	YouTube	11.62%
3	Generic QUIC	5.88%
4	Disney+	4.49%
5	TikTok	3.93%
6	HTTP Media Stream	3.72%
7	Playstation Downloads	2.95%
8	Xbox Live	2.91%
9	Facebook	2.87%
10	Amazon Prime	2.83%

б

UPSTREAM TRAFFIC

	Application	Total Volume
1	Netflix	8.78%
2	HTTP Media Stream	6.89%
3	YouTube	5.90%
4	Generic Web Browsing	3.85%
5	HTTP download	3.70%
6	Generic QUIC	3.43%
7	Generic Messaging	3.17%
8	Disney+	2.98%
9	Hulu	2.79%
10	Facebook	2.67%

в

Рисунок 1.2. Рейтинг компаній за загальним об’ємом трафіку (а), об’ємом завантаження (б) та об’ємом вивантаження (в) [4]

Основною спільною рисою цих систем є необхідність обробляти великі об'єми запитів у реальному часі, забезпечуючи високий рівень доступності, надійності та відгуку на зміни в умовах функціонування. Великі компанії забезпечують обробку дуже великих обсягів трафіку завдяки використанню інноваційних технологій і архітектурних рішень, що дозволяють досягти високої масштабованості та відмовостійкості.

Такі гіганти як Netflix, YouTube, Disney+, TikTok, Facebook, Xbox Live, Playstation Downloads і Amazon Prime здатні обробляти більше 52,65 % всього інтернет-трафіку, що генерується лише 10 застосунками, завдяки комплексному впровадженню сучасних рішень.

Основними підходами, які використовують ці компанії, є:

1. Розподілені архітектури і мікросервісна організація.

2. Використання мережевих технологій і контент-доставки.

3. Оптимізація мережевих протоколів і застосування сучасних алгоритмів балансування навантаження.

4. Постійний моніторинг, аналіз і адаптивне масштабування.

Одним із основних підходів у проєктуванні, що передбачає поділ застосунку на незалежні сервіси, є мікросервісна архітектура (рис. 1.3).

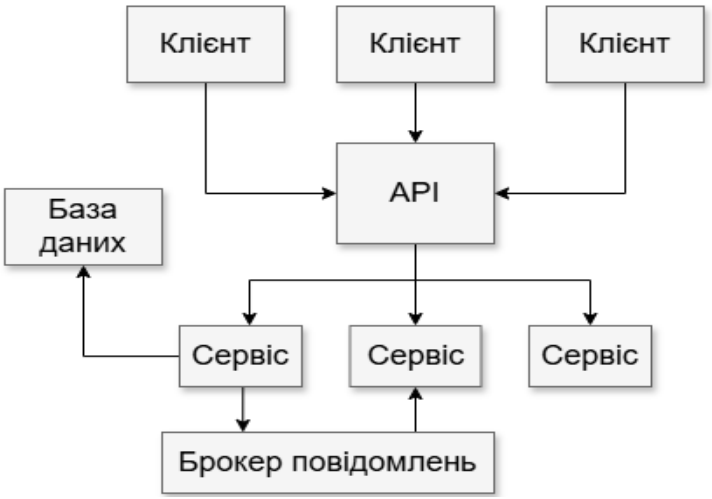

Рисунок 1.3. Приклад мікросервісної архітектури високонавантаженої системи

З рис. 1.3 видно, що сервіси можуть спілкуватися між собою за допомогою брокера повідомлень і мати доступ до бази даних, шо дозволяє кожному з них функціонувати без використання завчасно збережених даних.

Розподіл функціональності між незалежними сервісами створює передумови для досягнення оптимальної продуктивності і адаптивності системи, що дозволяє значно покращити стабільність роботи застосунків, які функціонують у середовищах з високими вимогами до безперервності і затримок в умовах змінних навантажень. Саме тому високонавантажені системи часто побудовані з використанням мікросервісної архітектури.

Можлива мікросервісна бекенд-інфраструктура Netflix, що сформована на основі доступних звітів і блогів, якими діляться експерти галузі, показана на рис. 1.4.

Процес обробки одного запиту до серверної інфраструктури Netflix включає декілька ключових етапів, реалізованих на базі AWS. Після того як клієнт ініціює запит на відтворення (Playback), запит проходить через розподілювач навантаження AWS Load Balancer (ELB), який приймає та перенаправляє вхідні запити до сервісу API Gateway, розгорнутого на AWS EC2.

Для ефективного управління запитами через AWS ELB Netflix використовує власну розробку – Zuulx, що забезпечує розширені можливості маршрутизації, моніторингу трафіку, безпеки та збереження даних. Цей компонент також сприяє підвищенню надійності системи, усуваючи потенційні точки відмови.

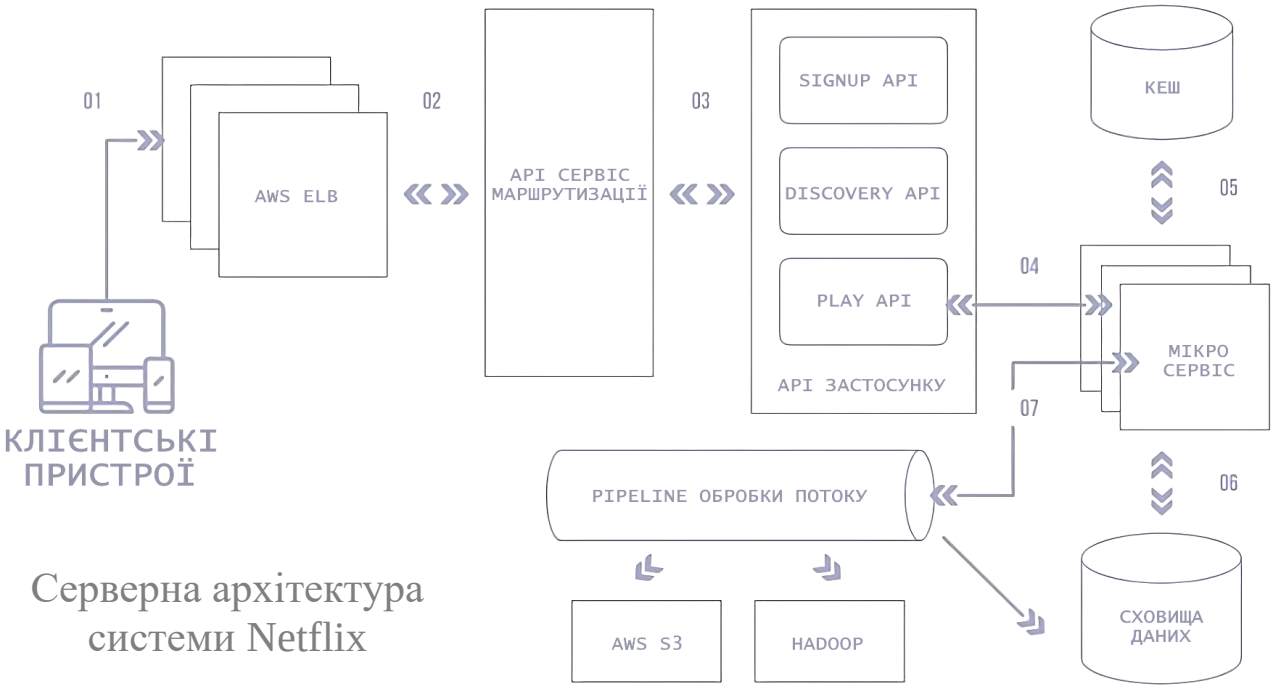

Рисунок 1.4. Схема серверної архітектури системи Netflix [5]

Далі запит передається до основного бізнес-логічного рівня, представленого Application API. У випадку запиту на відтворення контенту задіюється Play API, що функціонує в межах API Gateway Service. Для інших запитів, таких як автентифікації користувача чи перевірки підписки, передбачені відповідні API, що розгорнуті в межах Application API. Play API взаємодіє з низкою мікросервісів, які відповідають за виконання запиту. У випадку відтворення відео залучаються такі мікросервіси, як Playback Apps, Steering і Cache-Control, що забезпечують оптимальну роботу сервісу та керування кешуванням контенту.

Ключовим елементом забезпечення стійкості мікросервісної архітектури Netflix є розроблена компанією система ізоляції мікросервісів Hystrix, що зменшує кількість відмов і підвищує стійкість системи. Цей компонент дає змогу кожному мікросервісу працювати автономно, що знижує вплив потенційних збоїв у суміжних сервісах і підвищує ймовірність успішного виконання запитів.

Окрім обробки запитів, мікросервіси Netflix відстежують історію взаємодії і активність користувачів, передаючи ці дані до Stream Processing Pipeline. Це забезпечує реальну персоналізацію контенту і генерацію рекомендацій у режимі реального часу.

Оброблені дані передаються у сховища для подальшої аналітики за допомогою технологій AWS S3, Hadoop HDFS і Cassandra, що забезпечують ефективне управління великими масивами інформації. При цьому кожен сервіс системи, що має мікросервісну архітектуру (рис. 1.4), може мати власну кількість реплік для збільшення пропускної здатності системи і підвищення її здатності обробляти більші об’єми даних. Це означає, що вчасне масштабування забезпечує високонавантаженим розподіленим системам здатніість до адаптації згідно зі змінами у навантаженні та доступності ресурсів. Таким чином, Netflix реалізує масштабовану, стійку архітектуру, що здатна обробляти мільйони запитів в секунду, забезпечуючи високу продуктивність і надійність обслуговування користувачів [5].

Мережеві технології і контент-доставка дозволяють розміщувати кешовані копії контенту по всьому світу, зменшуючи затримки і навантаження на центральні сервери. Оптимізація мережевих протоколів і застосування таких алгоритмів балансування навантаження, як Generic QUIC і HTTP Media Stream, дозволяють ефективно управляти потоками даних і швидко реагувати на зміни трафіку.

Моніторинг у реальному часі є важливим елементом ефективного управління високонавантаженими розподіленими системами. Інструменти на кшталт Prometheus і Grafana дозволяють контролювати основні метрики їх продуктивності і швидко виявляти потенційні проблеми. Моніторинг, аналіз, адаптивне масштабування і візуалізація даних забезпечує можливість відстежувати показники систем і здійснювати проактивні дії щодо масштабування ресурсів у режимі реального часу. Це дозволяє зменшувати витрати і підвищувати продуктивність [6]. Автоматизовані інструменти тестування і CI/CD-пайплайни зменшують ризик людських помилок і підвищують ефективність процесу розгортання.

Успішне впровадження високонавантажених розподілених систем також передбачає ретельне тестування, при чому:

– навантажувальне тестування дає змогу оцінити поведінку системи за стандартних умов роботи;

– стрес-тестування визначає межі її продуктивності в умовах екстремальних навантажень:

– тестування масштабованості надає розуміння щодо здатності системи адаптуватися до мінливої кількості користувачів без втрати продуктивності.

Вчасне (ефективне) масштабування запобігає:

– зайвим витратам в разі внаслідок простою ресурсів при занадто ранньому масштабуванні;

– погіршенню якості сервісу через перевантаження при занадто пізньому масштабуванні.

Таким чином вибір відповідного підходу до балансування між точністю прогнозу і оперативністю реагування на зміну навантаження є важливим фактором забезпечення ефективності системи, що першою чергою потребує визначення часу запуску і вимикання додаткових ресурсів, щоб:

1) максимально згладити пікове навантаження;

2) не тримати зайвих потужностей довше необхідного.

1.2. Надійність і ефективність високонавантажених розподілених систем

Надійність високонавантажених розподілених систем є однією з ключових характеристик їхньої стабільної роботи, оскільки такі системи мають залишатися функціональними навіть за умов динамічних змін навантаження, локальних збоїв окремих компонентів або несподіваних втрат доступності. Зі зростанням вимог до безперервного обслуговування і мінімізації часу простою все більшу роль відіграють механізми автоматичного виявлення та усунення відмов, адаптивного балансування навантаження та реплікації критичних даних і обчислень.

Відмовостійкість є критично важливим аспектом надійності розподілених систем, а мікросервісна архітектура надає переваги у плані ізоляції збоїв та підтримки безперервної роботи навіть у випадку часткових відмов. Водночас її ефективність значною мірою залежить від правильної побудови механізмів моніторингу та автоматизації управління відмовами, без яких мікросервісні системи можуть стати менш передбачуваними та складнішими для діагностики проблем.

У [7] проведено аналіз основних архітектурних рішень, що спрямовані на підвищення надійності розподілених обчислень, зокрема розглянуто такі підходи, як мультизонова реплікація, автоматизовані механізми збереження стану, а також алгоритми перенаправлення запитів у разі збою одного або декількох вузлів системи. Особливу увагу приділено використанню механізмів реплікації, що дозволяють зберігати дублікати критично важливих даних у кількох незалежних дата-центрах або серверах, забезпечуючи їхню доступність навіть у разі виходу з ладу первинного вузла. Реалізація таких підходів неможлива без використання ефективних механізмів розподіленої узгодженості, які забезпечують коректну синхронізацію стану даних між кількома незалежними серверами.

У [8] розглянуто теоретичні основи таких алгоритмів, зокрема Paxos і Raft, які є одними з найбільш широко використовуваних протоколів досягнення узгодженості у розподілених середовищах. Алгоритм Paxos забезпечує стійкість до відмов шляхом використання консенсусу серед більшості вузлів кластера, що дозволяє приймати узгоджені рішення навіть у разі втрати зв’язку з окремими серверами. Raft є більш практичним варіантом консенсусного алгоритму, який спрощує імплементацію розподілених систем, оскільки включає чітко визначені ролі лідера і підлеглих вузлів, що мінімізує ризик конфліктів при зміні стану даних. Аналіз цих алгоритмів у контексті їхньої застосовності до високонавантажених систем демонструє, що вони є основою для забезпечення цілісності та доступності даних навіть у випадках часткових відмов серверів або втрати мережевих з'єднань.

Досягнення високої надійності високонавантажених розподілених систем потребує комплексного підходу, що включає використання автоматизованих механізмів реплікації, алгоритмів узгодженості та динамічного балансування навантаження, що дає змогу мінімізувати негативні наслідки відмов окремих компонентів і забезпечити стабільну роботу критично важливих сервісів у масштабованих середовищах.

У дослідженні [9] підкреслюється, що одним із фундаментальних критеріїв надійності високонавантажених розподілених систем є відмовостійкість. Цей критерій забезпечує здатність системи продовжувати коректне функціонування

навіть у разі виходу з ладу окремих компонентів або втрати доступу до певних обчислювальних вузлів. Це особливо важливо для критично важливих сервісів, коли збій може призвести до значних фінансових і операційних втрат.

Одним із підходів до підвищення відмовостійкості є застосування мікросервісної архітектури, що розглядається у [10] як ефективний засіб мінімізації наслідків відмови окремих компонентів. Завдяки тому, що мікросервіси працюють незалежно один від одного і взаємодіють через стандартизовані API, вихід з ладу окремого сервісу не спричиняє критичного збою всієї системи, а лише впливає на конкретну функціональність. Це дозволяє забезпечити гнучку ізоляцію відмов, що є ключовим фактором для досягнення високої надійності.

Проте, як зазначено у [11], впровадження мікросервісної архітектури створює нові виклики у забезпеченні надійності, зокрема необхідність ефективної оркестрації і моніторингу компонентів. Збільшення кількості незалежних мікросервісів у розподіленій системі ускладнює контроль за їхньою взаємодією, оскільки без належного інструментарію для спостереження за системним станом відмова окремих сервісів може бути не одразу виявлена. Крім того, розподілений характер мікросервісних додатків може створювати системні аномалії, такі як каскадні відмови або часткові деградації продуктивності, які складно виявити без детального аналізу.

В [11] також наголошується, що традиційні підходи до моніторингу, які працювали для монолітних систем, є недостатньо ефективними у мікросервісних середовищах, і тому необхідне використання таких сучасних рішень, як розподілені трейсинг-системи (OpenTelemetry), інтелектуальні механізми балансування навантаження та автоматизовані механізми відновлення (self-healing).

Дослідження [12] демонструє, що рівень надійності розподілених систем значною мірою визначається їхньою здатністю до відновлення після збоїв. Однією з ключових стратегій для забезпечення безперервності роботи є використання толерантності до збоїв, що дозволяє мінімізувати наслідки відмов окремих компонентів шляхом автоматичного перезапуску або переміщення навантаження на резервні ресурси. У розподілених системах, що працюють на основі контейнеризації, таких як Kubernetes, широко застосовуються алгоритми самовідновлення, які

відстежують стан контейнерів та автоматично перезапускають їх у разі збоїв або порушення очікуваних показників продуктивності. Це дозволяє відновлювати сервіси без необхідності ручного втручання адміністратора, що критично важливо для систем із високими вимогами до доступності.

У [13] розглянуто додатковий рівень підвищення надійності за допомогою автоматичної балансировки навантаження, яка сприяє рівномірному розподілу трафіку між вузлами та швидкому реагуванню на несправності. Зокрема, такі рішення, як Envoy і Istio, використовують розподілені проксі-сервери та механізми контролю стану сервісів для виявлення відмов і перенаправлення трафіку на найближчі доступні ресурси. Завдяки такому підходу можливо не лише автоматично компенсувати відмови окремих вузлів, але й оптимізувати маршрутизацію запитів, що знижує ризик перевантаження певних компонентів системи. Це забезпечує більш рівномірний розподіл навантаження та підвищує загальну стабільність розподіленої інфраструктури.

Ще одним ключовим критерієм надійності розподілених систем є час відновлення після збоїв (MTTR), що характеризує швидкість повернення системи до стабільного стану після виникнення аварійної ситуації. Високе значення MTTR свідчить про тривалий період простою, що може негативно позначитися на рівні доступності сервісів та призвести до фінансових втрат, особливо у високонавантажених системах із жорсткими вимогами до SLA. Тому зменшення часу MTTR є критично важливим завданням у проєктуванні надійних хмарних інфраструктур та мікросервісних архітектур.

У [14] проведено детальний аналіз методів скорочення часу відновлення, серед яких автоматизовані механізми аварійного відновлення, активне резервування компонентів і використання cloud failover-стратегій. В цій роботі показано, що застосування автоматизованого відновлення на рівні контейнерної оркестрації (Kubernetes self-healing) може зменшити середній час простою сервісів на 30–50%, порівняно з традиційними ручними підходами. В [14] також показано, що механізми гарячого резервування (hot standby) дозволяють практично миттєво перемикати навантаження на резервні вузли, що суттєво знижує вплив відмов на продуктивність.

У [15] розглядається використання розподіленого журналювання змін та технологій snapshotting, які дозволяють зменшити затримку при відновленні даних у розподілених середовищах. Журналювання змін забезпечує послідовне збереження критичних операцій у реальному часі, що у разі збою дає змогу відновити систему до останнього узгодженого стану без необхідності повторного виконання всіх операцій. Snapshotting, у свою чергу, дозволяє періодично зберігати повні знімки стану системи, що значно прискорює процес відновлення у разі критичних відмов. Авторами [15] доведено, що використання цих технологій у великих хмарних сховищах і розподілених базах даних дозволяє скоротити час відновлення на 40–60% порівняно з традиційними методами резервного копіювання.

Таким чином, для ефективного зниження MTTR доцільно використовувати комплексний підхід, що включає автоматизовані механізми відновлення, резервування на рівні інфраструктури, журналювання змін та періодичне створення знімків системи. Поєднання цих методів дає змогу досягти високої надійності та безперервності роботи сервісів, що є критично важливим для сучасних високонавантажених розподілених систем.

Ще одним із ключових критеріїв надійності високонавантажених розподілених систем є узгодженість даних, яка визначає рівень відповідності стану даних між усіма вузлами в системі. Узгодженість безпосередньо впливає на цілісність і коректність роботи сервісів, особливо в умовах розподілених транзакцій та паралельного доступу.

Фундаментальне обмеження для будь-якої розподіленої системи, що працює у ненадійній мережі формулює CAP-теорема [16]: не можливо одночасно забезпечувати консистентність (Consistency), доступність (Availability) та стійкість до мережевих розділень (Partition Tolerance) системи. Це означає, що при виникненні розподіленого збою система змушена вибирати між консистентністю та доступністю, оскільки неможливо гарантувати їх одночасне збереження. Умова CAP-теореми призводить до потреби вибору між двома основними підходами до узгодженості даних у сучасних розподілених системах ACID (Atomicity, Consistency, Isolation, Durability) і BASE (Basically, Available, Soft state, Eventually consistent).

Підхід ACID є типовим для традиційних реляційних баз даних, де пріоритетом є остаточну узгодженість даних навіть за рахунок доступності. Це гарантує, що всі транзакції є атомарними та ізольованими, а після їх виконання дані залишаються в узгодженому стані. Натомість підхід BASE, що використовується в багатьох NoSQL-системах (Cassandra, DynamoDB), відмовляється від остатоної консистентності на користь доступності і стійкості до мережевих розділень, допускаючи остаточну узгодженість, коли дані можуть бути розсинхронізоваі тимчасово, але зрештою всі вузли приходять до узгодженого стану.

Вибір між ACID і BASE залежить від конкретних вимог застосунку. Фінансові системи та критично важливі транзакційні сервіси потребують суворої узгодженості ACID для забезпечення коректності операцій, тоді як системи кешування, соціальні мережі та аналітичні платформи часто використовують BASE, оскільки швидкість обробки запитів і доступність даних є пріоритетними. У реальних сценаріях такі сучасні розподілені бази даних, як Google Spanner, CockroachDB і Amazon DynamoDB, поєднують обидва підходи, забезпечуючи слабко узгоджену (weak consistency) або конфігуровану (tunable consistency) модель, яка дозволяє користувачу самостійно балансувати між доступністю та консистентністю залежно від потреб бізнес-логіки.

Оцінка надійності високонавантажених розподілених систем є критичною для забезпечення безперебійної роботи, особливо в умовах динамічного масштабування, високої конкуренції за ресурси та можливих відмов окремих компонентів. Однак, надійність сама по собі не є достатнім показником якості роботи системи, оскільки її необхідно розглядати у взаємозв’язку з продуктивністю та ефективністю використання ресурсів.

Баланс між надмірною відмовостійкістю та економічною доцільністю є одним із ключових задач у проектуванні сучасних розподілених обчислювальних середовищ. З одного боку, надмірне дублювання даних, агресивні механізми реплікації і стратегічне резервування ресурсів можуть забезпечити високу надійність, але водночас призводять до значних операційних витрат і зниження загальної продуктивності. З іншого боку, надмірне зосередження на оптимізації критеріїв

швидкодії і витрат може призвести до збільшення ризику збоїв, що негативно позначається на стабільності роботи системи. Тому наступним важливим складовою аналізу є дослідження метрик ефективності, які дозволяють об'єктивно оцінити продуктивність розподілених систем та знайти компроміс між їхньою стійкістю, швидкодією та економічною доцільністю.

Оцінка ефективності високонавантажених розподілених систем є складним і багатовимірним процесом, що вимагає детального аналізу різних метрик. Для забезпечення всебічної оцінки ефективності необхідно враховувати спектр характеристик, які впливають на продуктивність, надійність і економічну доцільність використання обчислювальних ресурсів (рис. 1.5).

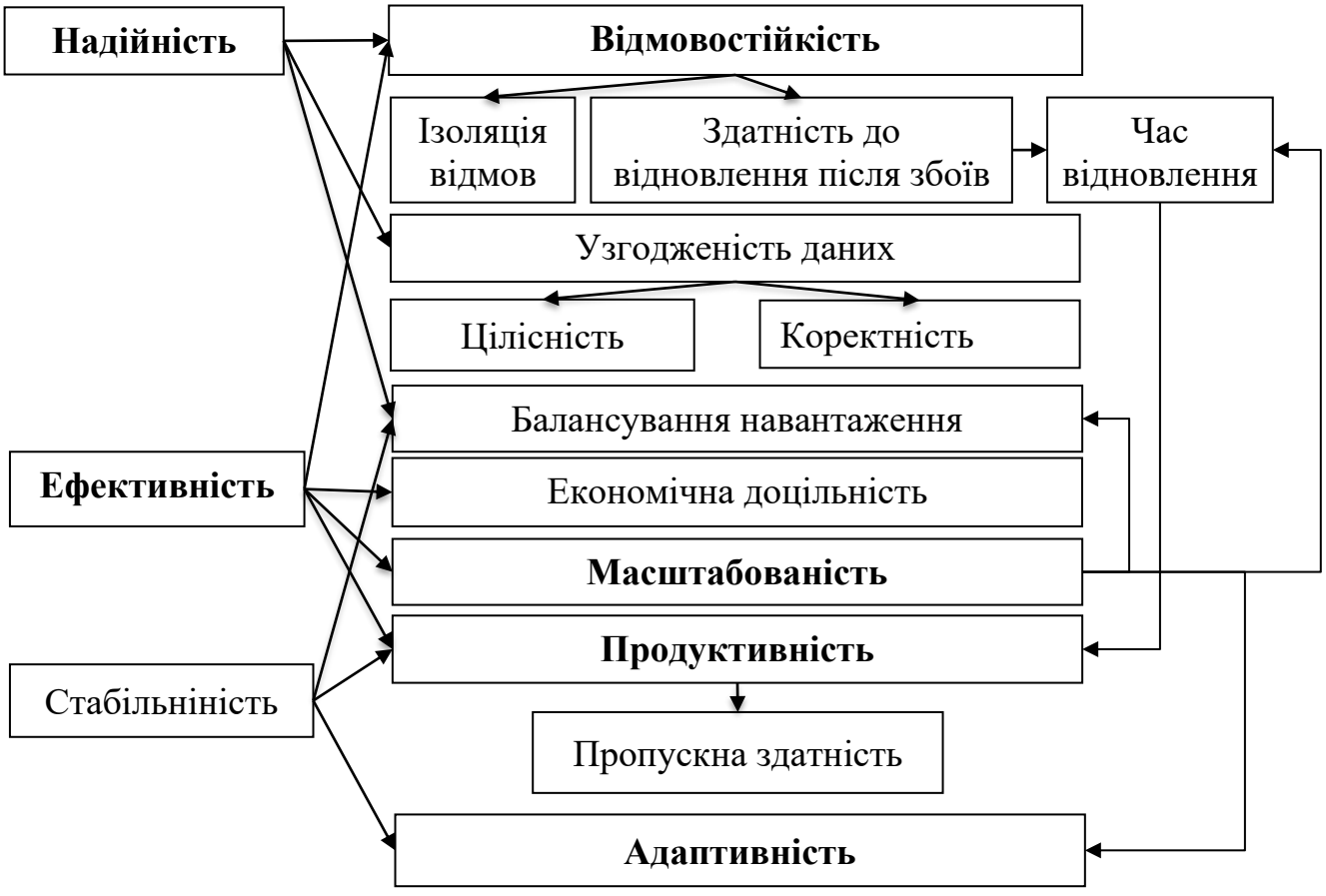

Рисунок 1.5. Співвідношення метрик роботи системи

У [17] запропоновано класифікацію ключових метрик ефективності, що відіграють важливу роль у забезпеченні факторів стабільної та ефективної роботи

системи: продуктивність, відмовостійкість, економічна ефективність (cost efficiency), масштабованість.

Метрики продуктивності охоплюють показники, що характеризують швидкість обробки запитів, пропускну здатність та середній час відгуку. Вони є критичними для оцінки можливостей системи щодо роботи під навантаженням і впливають на кінцевий користувацький досвід. Масштабованість визначає здатність системи адаптуватися до змін у навантаженні шляхом горизонтального або вертикального масштабування, що є необхідним для підтримки стійкої продуктивності в умовах динамічного розподілу обчислювальних завдань.

Відмовостійкість характеризує здатність системи зберігати працездатність при збоях окремих компонентів, включаючи механізми реплікації, автоматичного відновлення та балансування навантаження. Ця характеристика є визначальною для критичних сервісів, де непередбачені відмови можуть спричинити значні фінансові або репутаційні втрати. Економічна ефективність відображає баланс між споживанням обчислювальних ресурсів та досягнутою продуктивністю. Оптимізація витрат у хмарних обчисленнях і розподілених середовищах є ключовим завданням, особливо з урахуванням динамічного характеру робочих навантажень та необхідності автоматизованого керування ресурсами.

Оцінка ефективності високонавантажених розподілених систем вимагає комплексного підходу, що поєднує аналіз продуктивності, адаптивності до змінного навантаження, стійкості до збоїв і економічної доцільності використання обчислювальних ресурсів. Нижче розглядаються дослідження з розроблення оптимальних методик для забезпечення рівноваги між цими параметрами, що сприятиме підвищенню надійності та ефективності сучасних розподілених архітектур.

Одним із ключових показників ефективності високонавантажених розподілених систем є пропускна здатність (throughput), яка визначає максимальну кількість запитів або операцій, що можуть бути виконані системою за одиницю часу. Вона є важливим фактором продуктивності, оскільки безпосередньо впливає на

здатність системи обробляти великі обсяги даних і підтримувати стабільну роботу під значним навантаженням.

У [18] зазначено, що пропускна здатність є особливо критичною для мікросервісних архітектур і розподілених баз даних, оскільки їхня робота передбачає обробку великої кількості одночасних запитів, що можуть створювати значне навантаження на обчислювальні ресурси. Якщо система не має ефективного механізму масштабування, то при збільшенні навантаження можливе перевантаження серверів, що призводить до зростання часу відгуку та втрати продуктивності.

У [19] досліджено проблему підтримки високої пропускної здатності в умовах змінного навантаження. Запропонований підхід базується на адаптивному балансуванні навантаження, яке передбачає рівномірний розподіл запитів між доступними обчислювальними вузлами. Це дозволяє уникати ситуацій, коли одні вузли перевантажуються, тоді як інші залишаються недостатньо задіяними. Адаптивне балансування здійснюється за допомогою алгоритмів динамічного розподілу запитів, які враховують поточне навантаження та прогнозують подальший розподіл обчислювальних ресурсів. У результаті така стратегія забезпечує оптимальне використання ресурсів, запобігає утворенню вузьких місць і сприяє підвищенню загальної ефективності системи.

Залежно від архітектури системи і характеру навантаження, ефективність пропускної здатності також може покращуватися шляхом оптимізації механізмів кешування, партиціювання даних та паралельної обробки запитів. У деяких випадках використання таких методів масштабування, як реплікація даних або автоматичне горизонтальне масштабування, дозволяє збільшити пропускну здатність без значного зростання часу відгуку. Проте, як показано у [19], досягнення оптимального рівня пропускної здатності вимагає врахування низки параметрів, включаючи архітектурні обмеження, характеристики апаратного забезпечення та програмні механізми управління навантаженням.

Пропускна здатність є визначальним критерієм продуктивності розподілених систем, і її підтримка на належному рівні вимагає комплексного підходу, що включає адаптивне балансування навантаження, оптимізацію використання ресурсів та

ефективне горизонтальне масштабування. Майбутні дослідження мають бути спрямовані на вдосконалення алгоритмів прогнозування навантаження та адаптації систем до динамічних умов експлуатації для забезпечення максимальної ефективності розподілених обчислювальних середовищ.

Середній час відгуку є одним із ключових параметрів ефективності високонавантажених розподілених систем, що визначає часовий інтервал між моментом надсилання запиту користувачем або внутрішнім сервісом та отриманням відповіді від системи. Як показано у [20], цей показник має вирішальне значення для інтерактивних сервісів, таких як веб-додатки, системи онлайн-банкінгу та потокові платформи, де навіть незначні затримки можуть суттєво впливати на користувацький досвід і рівень задоволеності. У дослідженнях, що проведені компанією Google, зазначено, що затримка понад 200 мс у часі відгуку пошукової системи може призвести до значного зниження кількості запитів від користувачів і, відповідно, до втрати доходів рекламних сервісів.

У роботі [21] детально проаналізовано методи оптимізації середнього часу відгуку у високонавантажених системах, включаючи кешування запитів, асинхронну обробку даних і використання балансувальників навантаження. Зокрема, було показано, що застосування алгоритмів кешування на рівні серверів зменшує затримки обробки повторюваних запитів у середньому на 40%, що суттєво покращує продуктивність у сценаріях з високим рівнем однотипних запитів. Крім того, використання асинхронної обробки дозволяє зменшити час очікування в розподілених системах, у яких операції введення-виведення можуть спричиняти суттєві затримки, особливо під час роботи з базами даних або зовнішніми API.

Проте, як зазначено у [22], середній час відгуку не завжди є достатньо інформативним показником оцінки продуктивності системи, оскільки він усереднює запити, не враховуючи крайові випадки. Зокрема, у масштабованих середовищах часто виникають ситуації, коли більшість запитів обробляється швидко, але невелика частина може мати значно більші затримки через навантаження на окремі вузли системи, мережеві обмеження або нестачу ресурсів у певний момент часу. Це явище, відоме як хвостова латентність, має критичне значення для таких сервісів реального

часу, як фінансові біржі або ігрові хмарні платформи, де навіть поодинокі випадки надмірних затримок можуть спричинити серйозні наслідки. Таким чином, для комплексної оцінки ефективності розподілених систем необхідно враховувати не лише середній час відгуку, але й показники хвостової латентності, що дозволяють точніше визначити стабільність роботи системи під змінними навантаженнями.

Хвостові затримки є критичним аспектом продуктивності розподілених високонавантажених систем, що визначає час обробки найповільніших запитів у системі. На відміну від середнього часу відгуку, який може свідчити про загальну продуктивність системи, хвостові затримки фокусуються на найгірших сценаріях роботи окремих запитів, що можуть суттєво впливати на загальну якість обслуговування (QoS). У [23] зазначено, що в умовах масштабованих мікросервісних архітектур, де кожен запит може проходити через кілька незалежних сервісів або вузлів, навіть невелика частка повільних запитів може спричинити каскадну деградацію продуктивності. Це особливо критично для фінансових і телекомунікаційних систем, де гарантована швидкість обробки даних є ключовою вимогою до безперебійного надання послуг. Так у біржових торговельних системах навіть незначне збільшення часу обробки транзакцій може призвести до втрат мільйонів доларів через упущені можливості, тоді як у телекомунікаційних сервісах затримки у маршрутизації запитів можуть викликати перебої у з’єднанні або зниження якості обслуговування абонентів.

У [24] досліджується вплив механізмів інтелектуального кешування і адаптивної маршрутизації запитів на зменшення хвостових затримок. Автори демонструють, що стандартні механізми кешування часто орієнтовані на загальне зниження середнього часу відгуку, однак не завжди ефективно працюють у випадках пікових навантажень. Використання адаптивного кешування дозволяє зменшити навантаження на критичні вузли системи, заздалегідь передбачаючи запити, що мають високий ризик тривалої обробки. Крім того, застосування алгоритмів розподілу запитів, які враховують поточний стан серверів і їх завантаження, допомагає мінімізувати ймовірність виникнення вузьких місць, що можуть спричинити значні сплески хвостових затримок. Це є важливим фактором оптимізації продуктивності розподілених систем,

особливо для сервісів із високими вимогами до швидкодії. Використання поєднання адаптивного кешування, інтелектуального балансування навантаження та методів попереднього прогнозування дозволяє значно зменшити ризики надмірних затримок і покращити загальну ефективність системи.

Масштабованість системи є одним із ключових показників її ефективності, оскільки визначає здатність адаптуватися до змінного навантаження без втрати продуктивності і стабільності роботи. Горизонтальне масштабування передбачає додавання нових обчислювальних вузлів до системи, що забезпечує її лінійне розширення та підвищення пропускної здатності без змін у конфігурації окремих серверів [25]. Такий підхід широко застосовується в розподілених мікросервісних архітектурах, оскільки дозволяє досягти високої відмовостійкості та уникати єдиної точки відмови. Проте, ефективність горизонтального масштабування значною мірою залежить від реалізації балансування навантаження та міжвузлової комунікації, що може створювати додаткові затримки у великих кластерах.

Вертикальне масштабування, на відміну від горизонтального, передбачає збільшення обчислювальної потужності окремого вузла шляхом додавання процесорних ядер, оперативної пам’яті або інших ресурсів. У [26] розглянуто ефективність цього підходу в умовах обмежених обчислювальних середовищ, таких як монолітні додатки або бази даних, що вимагають збереження цілісності процесів на одному сервері. Перевагою вертикального масштабування є зменшення накладних витрат на координацію вузлів, оскільки всі ресурси зосереджені в межах однієї фізичної або віртуальної машини. Проте цей метод має фізичні обмеження, оскільки ресурси сервера не можуть зростати нескінченно, а модернізація обладнання часто супроводжується високими витратами. Сучасні високонавантажені розподілені системи зазвичай використовують комбіновані підходи, що поєднують горизонтальне і вертикальне масштабування залежно від поточного стану навантаження та економічної доцільності.

У [26] аналізуються методи динамічного балансування навантаження та авто-масштабування ресурсів у Kubernetes, що дозволяють автоматично адаптувати конфігурацію системи відповідно до змінних умов експлуатації. Kubernetes Horizontal

Pod Autoscaler (HPA) реалізує реактивний підхід до горизонтального масштабування, збільшуючи або зменшуючи кількість Pod у кластері залежно від поточного завантаження CPU або інших метрик. Водночас Vertical Pod Autoscaler (VPA) дозволяє динамічно змінювати обсяг виділених ресурсів для окремих контейнерів, що забезпечує гнучке управління продуктивністю кожного компонента системи. Вибір між горизонтальним і вертикальним масштабуванням залежить від специфіки застосунку, архітектури системи та вимог до продуктивності. У реальних сценаріях найкращі результати часто досягаються завдяки гібридним стратегіям, які поєднують можливості обох підходів, забезпечуючи і швидке реагування на навантаження і оптимізацію використання обчислювальних ресурсів.

Окрім продуктивності та масштабованості, важливим фактором ефективності високонавантажених розподілених систем є коефіцієнт відмов (failure rate), який відображає частоту виникнення відмов окремих компонентів або системи загалом протягом певного періоду часу. Високий коефіцієнт відмов може суттєво знижувати доступність сервісів, призводити до втрати даних або збоїв у критичних бізнес-процесах. У [27] досліджено механізми реплікації даних та дублювання запитів, які дозволяють зменшити вплив відмов окремих вузлів у розподіленій інфраструктурі. Запропоновані алгоритми активного дублювання запитів у кількох дата-центрах дають змогу зменшити кількість помилок, що виникають через перевантаження або несправність серверів, забезпечуючи безперервну роботу критично важливих сервісів навіть у випадку втрати окремих вузлів.

Дослідження [28] показує, що для мінімізації негативного впливу відмов необхідне впровадження стратегій автоматичного відновлення після збоїв (self-healing strategies), які дозволяють швидко ідентифікувати проблемні вузли та здійснювати їхнє автоматизоване переведення у працездатний стан. Авторами розглянуто механізми автоматичного перезапуску контейнерів у Kubernetes, а також технології автоматизованого міграційного балансування навантаження, які забезпечують оперативне перенесення активних запитів на інші вузли без втрати доступності сервісів. Результати експериментів свідчать, що використання цих стратегій дозволяє зменшити середній час простою (MTTR) на 35–50%, підвищуючи

загальну відмовостійкість системи і її здатність до самовідновлення. Це є ключовим фактором для масштабованих хмарних платформ, де навіть короткочасне припинення роботи окремих сервісів може мати значний економічний вплив і спричиняти незадоволеність користувачів.

Ключовими показниками економічної ефективності є рівень використання ресурсів (resource utilization) і вартість обробки одного запиту (cost per request). Рівень використання ресурсів визначає, наскільки ефективно система застосовує доступні обчислювальні потужності, включаючи процесорний час, оперативну пам’ять, дисковий простір і мережеві ресурси. Високе використання ресурсів без перевантаження є показником оптимальної продуктивності, тоді як низьке використання свідчить про нераціональне резервування або неефективний розподіл обчислювальних потужностей.

У [29] проведено аналіз витрат у хмарних середовищах, який показав, що неоптимальне резервування ресурсів призводить до суттєвого зростання експлуатаційних витрат, особливо у випадках, коли система завжди утримує більше обчислювальних вузлів, ніж фактично потрібно для обробки поточного навантаження. Це особливо актуально для хмарних платформ, що працюють за моделлю pay-as-you-go, де провайдери виставляють рахунок на основі використаних ресурсів. Надмірне масштабування системи «про запас» без точного прогнозування реального навантаження може збільшити витрати на інфраструктуру на 30–50%, не забезпечуючи при цьому значного покращення продуктивності.

Робота [30] демонструє, що використання адаптивного масштабування з урахуванням прогнозованого навантаження дозволяє значно оптимізувати витрати на інфраструктуру без зниження продуктивності системи. Автори дослідження запропонували модель автоматичного масштабування, яка комбінує методи короткострокового прогнозування навантаження та динамічного балансування ресурсів. Результати експериментів показали, що такий підхід дозволяє знизити витрати на інфраструктуру в середньому на 25%, одночасно підтримуючи необхідний рівень продуктивності завдяки гнучкому коригуванню обсягу використовуваних обчислювальних ресурсів відповідно до змін навантаження в реальному часі.

Використання прогнозованого масштабування і алгоритмів адаптивного розподілу обчислювальних потужностей дозволяє мінімізувати надлишкові витрати, забезпечуючи при цьому достатній рівень обчислювальної продуктивності для задоволення поточного навантаження.

Аналіз наукових досліджень свідчить, що ефективність високонавантажених розподілених систем визначається балансом між продуктивністю, відмовостійкістю та економічною ефективністю.

Отже, вибір методу масштабування залежить від характеру навантаження, наявних обчислювальних ресурсів та вимог до часу реакції.

Вчасне (ефективне) масштабування запобігає:

– зайвим витратам в разі внаслідок простою ресурсів при занадто ранньому масштабуванні;

– погіршенню якості сервісу через перевантаження при занадто пізньому масштабуванні.

Таким чином вибір відповідного підходу до балансування між точністю прогнозу і оперативністю реагування на зміну навантаження є важливим фактором забезпечення ефективності системи, що першою чергою потребує визначення часу запуску і вимикання додаткових ресурсів, щоб:

1) максимально згладити пікове навантаження;

2) не тримати зайвих потужностей довше необхідного.

Досліджено основні переваги і обмеження, що визначають ефективність різних методів автоматичного масштабування в високонавантажених розподілених системах та зниження ризиків збоїв відмовостійкість, адаптивність.

1.3. Принципи і методи масштабування високонавантажених розподілених систем

Масштабованість системи є одним із ключових показників її ефективності, оскільки визначає здатність адаптуватися до змінного навантаження без втрати продуктивності і стабільності роботи.

Принципи масштабування високонавантажених розподілених систем поділяються на вертикальне і горизонтальне (рис. 1.6).

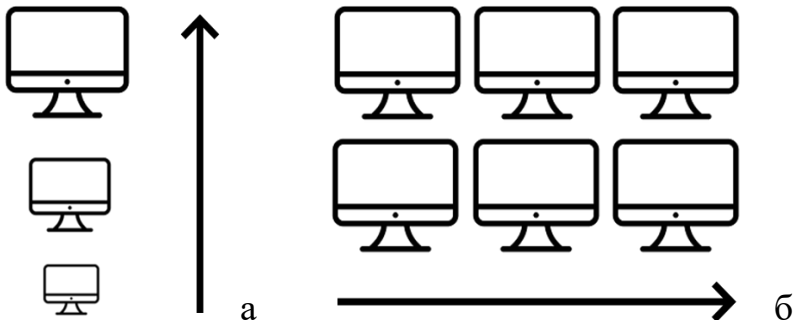

Рисунок 1.6. Вертикальне (а) та горизонтальне (б) масштабування

Вибір підходу (принципів і методів) масштабування системи залежить від специфіки застосунку.

Вертикальне масштабування передбачає збільшення обчислювальної потужності окремих серверів або віртуальної машини за рахунок додавання ресурсів (CPU, RAM, GPU). Це дозволяє зменшити накладні витрати на міжсерверну взаємодію, але має обмеження через фізичні ресурси обладнання. У базах даних, де важливим є низький час затримки доступу до даних, вертикальне масштабування часто є вигіднішим, тоді як горизонтальне масштабування забезпечує більшу гнучкість і адаптивність до навантаження в мікросервісних архітектурах.

При горизонтальному масштабуванні системи у відповідь на зростання навантаження відбувається додавання вузлів, що забезпечує безперебійну обробку запитів навіть у пікові моменти. Використання контейнеризації та оркестрації за допомогою Kubernetes значно полегшує керування ресурсами і спрощує автоматичне розгортання при зростанні навантаження. Це забезпечує високу гнучкість і дає змогу масштабувати окремі компоненти без зміни всієї системи. Таким чином, горизонтальне масштабування, на відміну від вертикального, реалізується шляхом додавання нових серверів або контейнерів, що дозволяє системі адаптуватися до стрімкого зростання навантаження і забезпечує кращу відмовостійкість і гнучкість, однак потребує балансування навантаження та синхронізації між вузлами [31-32].

Згідно з [31] в багатьох сучасних високонавантажених розподілених системах використовуються горизонтальні схеми масштабування, тому основною характеристикою високонавантажених розподілених систем є можливість горизонтального масштабування, що дозволяє збільшити обчислювальну потужність шляхом додавання нових вузлів без значного впливу на загальну продуктивність системи. Це підтверджується даними таких технологій, як Kubernetes, що забезпечує автоматичне масштабування контейнеризованих застосунків.

Методи управління ресурсами поділяють на статичні, реактивні, проактивні та гібридні. Реактивне управління ресурсами базується на моніторингу поточного стану системи і зміні конфігурації у відповідь на досягнення певних порогових значень. Ефективність реактивного підходу для хмарних систем, де рішення про збільшення чи зменшення ресурсів приймається на основі рівня використання CPU, пам’яті або мережевого трафіку досліджено в [32].

Ці рішення розраховані на системи, масштабування яких залежить від значних обсягів даних і складної архітектури, а витрати на інфраструктуру виправдовуються масштабами використання [33]. Статичне масштабування полягає у попередньому виділенні ресурсів за заздалегідь визначеними параметрами, що ефективне для передбачуваних навантажень. Однак, статичне масштабування призводить до перевитрат ресурсів у періоди низького навантаження або деградації продуктивності під час піків. Тому статичне масштабування погано працює в умовах динамічних змін [34].

Реактивне масштабування є найбільш простим у реалізації, проте воно не здатне запобігати перевантаженню, оскільки працює із затримкою, що може призводити до порушень SLA [35]. Іншим суттєвим недоліком реактивного масштабування є затримка у виконанні операцій, що може призвести до короткочасного зниження продуктивності або навіть відмов системи під час різких стрибків навантаження [36].

Прогнозоване масштабування на основі ML забезпечує високу точність при динамічних змінах навантаження, проте його складність і потреба у великих обчислювальних ресурсах обмежують сферу застосування [37].

Серед сучасних рішень автоматичного масштабування в хмарних системах виділяються сервіси, що запропоновані такими провідними платформами, як Google Cloud і Amazon Web Services (AWS).

AWS пропонує аналогічні підходи Predictive Scaling, що інтегрують ML для прогнозування пікових навантажень [38].

Google Cloud використовує підходи до масштабування, що базуються на передбаченні навантаження шляхом аналізу історичних даних з використанням машинного навчання (ML) для виявлення шаблонів і прогнозування майбутніх запитів [39].

У [40-42] запропоновано використання методів ML, таких як ARIMA та LSTM, для аналізу історичних даних навантаження та прогнозування піків.

Проактивне масштабування передбачає використання прогнозних моделей для передбачення майбутнього навантаження і завчасного розгортання додаткових ресурсів. В [43] показано, що проактивне масштабування суттєво знижує ймовірність перевантаження системи, особливо у випадках, коли прогнозні моделі демонструють високу точність.

Водночас, неточний прогноз може призвести до передчасного збільшення ресурсів і відповідно до перевитрат. Гібридні методи масштабування поєднують реактивні та проактивні механізми, що дозволяє компенсувати недоліки кожного з них. Прогнозоване масштабування на основі статистичного прогнозування менш затратний, але ефективний варіант, що працює у випадках стабільних циклічних змін і втрачає точність у нестабільних середовищах.

Підхід, де система виконує попереднє масштабування на основі прогнозу, але коригує кількість ресурсів у режимі реального часу відповідно до фактичних показників навантаження запропоновано в [44], а в [45] показано, що такі системи можуть зменшити загальні витрати на обчислювальні ресурси на 20–30% порівняно з традиційним реактивним підходом.

Основні переваги та обмеження, що визначають ефективність різних методів автоматичного масштабування в різних високонавантажених розподілених системах показано в табл. 1.1.

Таблиця 1.1.

Порівняльний аналіз методів автоматичного масштабування

Метод	Опис	Переваги	Недоліки	Найкраще використання
Реактивний	Реагує на досягнення порогових значень метрик у режимі реального часу	Простота реалізації, не потребує попереднього аналізу	Затримка реакції на раптові сплески, можливі порушення SLA	Системи з низькою вартістю затримки та передбачуваними сплесками навантаження
Прогнозований (на основі ML)	Використовує ML для прогнозування навантаження і масштабування на випередження	Висока точність для складних і динамічних робочих навантажень	Високі витрати на обчислення, складність впровадження	Великі динамічні системи з достатніми ресурсами та бюджетами
Прогнозований (статистичний)	Використовує статистичні моделі для прогнозування навантаження	Низька вартість, зрозумілість, підходить для передбачуваних шаблонів	Менш ефективне для сильно нестабільних чи нециклічних навантажень	Чутливі до витрат системи з стабільними та передбачуваними шаблонами навантаження
Гібридний	Поєднує прогнозування з реактивними механізмами для реального часу	Прогнозування на випередження і швидка реакція	Більша складність через інтеграцію кількох механізмів	Критичні системи, що вимагають точності і швидкої реакції

Найбільш універсальним підходом є гібридне масштабування, що поєднує проактивне прогнозування з реактивними механізмами, забезпечуючи баланс між точністю і швидкістю реагування, але водночас вимагає складнішої інтеграції і налаштувань. Тому ці дослідження зосереджуються на оптимізації гібридних методів, використанні штучного інтелекту для підвищення точності передбачень і розробці економічно ефективних моделей автоматизованого управління ресурсами.

Значним внеском в розвиток високонавантажених розподілених систем у контексті управління ресурсами є дослідження [46], яке пропонує гібридний підхід до автоматичного масштабування на основі прогнозування навантаження за часовими рядами. Автори [46] демонструють переваги інтеграції проактивних і реактивних підходів, що дозволяє ефективно реагувати на зміни навантаження та зменшувати витрати ресурсів.

В [47] запропоновано автоматизовану систему вибору оптимальних алгоритмів аналізу часових рядів, що знижує складність і підвищує ефективність процесу прогнозування, а в [48] запропропоновано фреймворк TempoScale, що використовує аналіз коротко- і довгострокових змін із застосуванням складних алгоритмів і нейромереж.

TempoScale аналізує історичні дані навантаження та проводить декомпозицію на різні функції для окремого прогнозування на короткостроковий і довгостроковий періоди (рис.1.7).

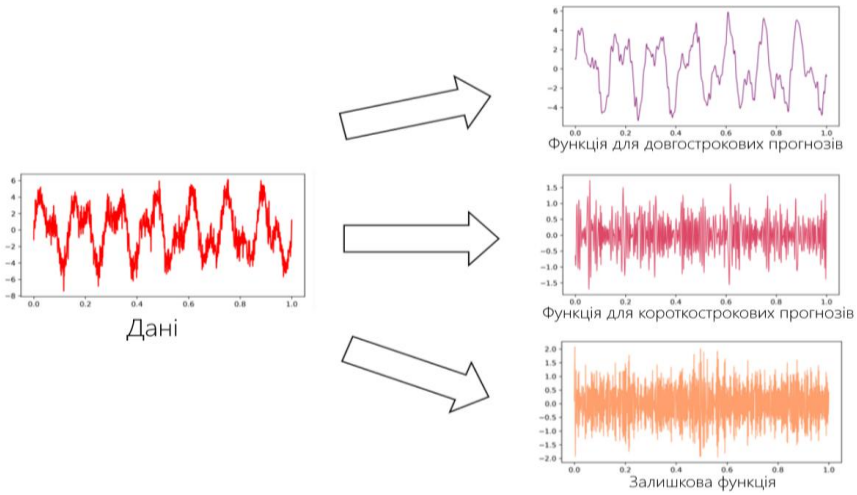

Рисунок 1.7. Модальна декомпозиція даних методом TempoScale [24]

Цей фреймворк підвищує продуктивність і стабільність системи, а також ефективно знижує витрати на ресурси, сприяючи сталому розвитку хмарних обчислень у різних галузях, використаний підхід демонструє ефективність у хмарних середовищах. Проте складність і орієнтація TempoScale на Kubernetes роблять його непридатним для систем, що використовують інші оркестратори.

У дослідженні [49] запропоновано підхід до автоматичного масштабування на основі OpenStack Monasca, що використовує методи прогнозування за допомогою нейронних мереж для передбачення динаміки ключових метрик і проактивного масштабування.

Цей підхід є оптимальним для систем, що використовують саме це хмарне рішення, оскільки полегшує впровадження і оптимізацію продуктивності. Проте, попри демонстрацію високої точності в хмарних середовищах, складність реалізації та залежність від інфраструктури OpenStack обмежують його застосування в локальних системах із простішими архітектурними вимогами.

В дослідженні [50] розглянуто застосування оптимізованої нейромережі LSTM для прогнозування короткострокових змін у високому енергоспоживанні. Попри високу точність результатів, що досягнута завдяки вдосконаленню моделі і застосуванню алгоритмів оптимізації, метод, що викормствує LSTM, значно залежить від складної попередньої обробки даних і високих обчислювальних ресурсів.

У статті [51] також запропоновано підхід до управління ресурсами на основі глибокого навчання, зокрема моделі LSTM. Проте його складність і орієнтація на хмарні сервіси з жорсткими вимогами до якості обслуговування (QoS) накладають серйозні обмеження на простір застосування цього підходу.

Загалом, аналіз останніх публікацій показує, що основні напрями досліджень зосереджені на розробці адаптивних методів масштабування, прогнозуванні навантаження та оптимізації обробки просторових даних. Застосування машинного навчання та нечіткої логіки дозволяє суттєво покращити показники продуктивності, знизити затримки у масштабуванні та оптимізувати використання обчислювальних ресурсів.

Проте залишається проблема узгодженості таких методів із реальними сценаріями експлуатації, оскільки навіть невеликі помилки у прогнозах можуть призводити до суттєвих фінансових втрат або нестачі ресурсів у критичні моменти. Тому дослідження, що спрямовані на розробку більш адаптивних та самонавчальних алгоритмів, які поєднуватимуть високу точність прогнозування з мінімізацією витрат та часу реакції на зміни навантаження, лишаються актуальними.

1.4. Постановка задачі

Основним завданням цього дисертаційного дослідження є розробка моделі адаптивного управління ресурсами високонавантажених розподілених систем, що дозволить ефективно реагувати на динамічні зміни навантаження, мінімізувати витрати ресурсів та забезпечити стабільну продуктивність системи.

Вирішення цього завданням передбачає:

1. Розробку математичної моделі динамічного масштабування ресурсів, та відповідного адаптивного нечіткого регресивного методу здатного до плавного та своєчасного керування ресурсами у високонавантажених умовах.
2. Розробку методу коригування моменту масштабування, що визначає найкращий момент зміни конфігурації ресурсів на підставі вихідних величин методу.
3. Моделювання навантаження розподілених систем, що включає різні типи навантаження (короткочасні піки, регулярні коливання, поступове зростання активності) та оцінку ефективності роботи розроблених методів.
4. Експериментальні дослідження та аналіз роботи запропонованих методів на реальних прикладах високонавантажених систем, таких як геоінформаційний модуль системи відновлення пошкоджених об'єктів, інтелектуальна система керування міським трафіком та аналітична система для моніторингу активності користувачів соціальних мереж.

Висновки до розділу 1

1. Аналіз сучасних високонавантажених розподілених систем показав, що їх ефективність і стійкість залежать від архітектурних рішень, механізмів управління ресурсами і вибору технологічного стеку. Визначено, що головними викликами у таких системах є баланс між продуктивністю, масштабованістю та відмовостійкістю, що вимагає оптимізації на рівні інфраструктури і на рівні програмного забезпечення.

Проєктування таких систем потребує визначення очікуваного навантаження, архітектурного планування та вибору технологій, що дозволяють забезпечити ефективне горизонтальне масштабування для уникнення затримок і надмірних витрат. Врахування цих факторів дає змогу досягти високої продуктивності, стійкості до відмов і оптимального використання ресурсів. Мікросервісна архітектура є ключовою концепцією побудови високонавантажених розподілених систем, оскільки забезпечує ефективне управління обчислювальними ресурсами, високу гнучкість масштабування та зниження ризиків збоїв. На відміну від монолітної, мікросервісну систему легше масштабувати та оптимізовувати.

2. Зясовано, що вчасне масштабування забезпечує високонавантаженим розподіленим системам здатність до адаптації згідно зі змінами у навантаженні та доступності ресурсів, а створення високонавантажених застосунків є ключовим викликом для розробників. _

3. Аналіз критеріїв надійності та метрик ефективності показав, що сучасні розподілені системи оцінюються за сукупністю параметрів, які включають продуктивність, відмовостійкість, узгодженість даних та економічну ефективність. Пропускна здатність, час відгуку та хвостові затримки визначають продуктивність системи, тоді як рівень відмовостійкості оцінюється через коефіцієнт відмов і середній час відновлення після збою. Масштабованість системи напряму впливає на її здатність адаптуватися до змін навантаження, а економічна ефективність вимірюється рівнем використання ресурсів і вартістю обробки запитів. Встановлено, що ефективне управління ресурсами і застосування методів адаптивного масштабування дозволяє знизити витрати без втрати продуктивності.

4. Аналіз останніх досліджень високонавантажених систем показав, що цю роботу доцільно зосередити на оптимізації обробки просторових даних і розробці адаптивних методів прогнозування навантаження та масштабування, які стануть теоретичною базою для подальшої розробки методів підвищення продуктивності та надійності висонавантажених систем спеціального призначення.

Основні наукові результати по цьому розділу опубліковані в працях [17].

РОЗДІЛ 2. МОДЕЛІ ТА МЕТОДИ АВТОМАТИЧНОГО МАСШТАБУВАННЯ ВИСОКОНАВАНТАЖЕНИХ СИСТЕМ

У розділі проаналізовано існуючі методи автоматичного масштабування високонавантажених систем і визначено необхідність адаптивного управління ресурсами в умовах аномальних і непередбачуваних навантажень; розглянуто вплив динамічних змін у запитах на ефективність масштабування та виявлено, що жорсткі порогові значення можуть призводити до неефективного розподілу ресурсів, що підтверджує доцільність використання більш гнучких стратегій масштабування; досліджено реактивні методи автоматичного масштабування, серед яких метод порогових правил, автоматичне масштабування на основі теорії черг та адаптивні реактивні алгоритми; показано, що ці методи є ефективними у випадках різких змін навантаження, однак вони мають певні обмеження, пов’язані з запізненням у реакції на зміну системних параметрів; обґрунтовано необхідність комбінування реактивних методів із проактивними підходами для підвищення ефективності управління ресурсами; розглянуто можливості застосування проактивних методів автоматичного масштабування, зокрема машинного навчання, розширеного навчання з підкріпленням, нечіткої логіки та аналізу часових рядів; показано, що використання проактивних методів дозволяє передбачати майбутні навантаження та ініціювати масштабування завчасно, що зменшує ризик перевантаження системи; обґрунтовано необхідність розробки комбінованих стратегій, які б дозволяли поєднувати прогнозні підходи з механізмами негайного реагування на зміну навантаження.

2.1 Управління аномаліями та непередбачуваними навантаженнями

Профіль навантаження у системі може мати різний характер змін у часі. Умовно виділяють декілька типових патернів хвиль навантаження, що впливають на стратегію масштабування: короткочасні сплески, періодичні піки та трендові довготривалі зміни. Кожен тип хвиль має свої особливості (амплітуду коливань,

тривалість перебування на піку і частоту повторення пікових періодів), за якими визначається оптимальна стратегія масштабування (табл. 2.1).

Короткочасні сплески, різкі одноразові підйоми навантаження тривають від секунд до хвилин, після чого інтенсивність швидко повертається до попереднього рівня. Характерна риса цього патерну – висока амплітуда (значне перевищення середнього навантаження) при короткій тривалості. Частота повторення сплесків невисока або нерегулярна (радше поодинокі випадки, ніж частий цикл).

Короткочасні сплески створюють найбільший виклик, оскільки час на реакцію системи обмежений тривалістю піку. Таким чином пік може минути, перш ніж додаткові ресурси встигнуть запуститися. Це призведе до просідання продуктивності якщо масштабування відбувається занадто повільно [52].

Таблиця 2.1

Класифікація типів хвиль навантаження

Клас	Тип патерну	Амплітуда	Оптимальна стратегія масштабування
1	Короткочасні сплески	Висока – значно перевищує середній рівень	Швидке масштабування, гарячі резерви, безсерверні функції
2	Періодичні піки	Середня або висока, але прогнозована	Проактивне масштабування за розкладом або прогнозне
3	Трендові довготривалі зміни	Плавне наростання або плавний спад	Поступове нарощування ресурсів, адаптивне коригування

Періодичні піки зазвичай більш передбачувані, оскільки відома їхня частота повторення, приблизна амплітуда і тривалість. Наявність таких шаблонів дозволяє застосувати проактивне масштабування – планове додавання ресурсів перед очікуваним піком. Таким чином, при настанні піку система вже має достатню кількість ресурсів і не витрачає час на наздоганяння навантаження.

Окрім чітко виражених піків, навантаження може змінюватися поступово у вигляді тренду – монотонного зростання або спадання протягом тривалого періоду.

Це може бути постійне загальне підвищення базового рівня навантаження на декілька відсотків щотижня. Трендові зміни характеризуються тим, що кроки наростання відносно малі в кожен момент часу, але за великий інтервал часу накопичуються значні зміни навантаження [53]. У цілому довготривалий тренд легше прогнозувати, тому задача зводиться до вирішення коли саме додати наступний ресурс, але важливо відслідковувати тренд, щоб вчасно коригувати базовий розмір інфраструктури для масштабування.

Одне з ключових питань ефективного масштабування полягає у визначенні часу коли запускати (t_0) додаткові ресурси і коли їх вимикати (t_f), щоб:

1) максимально згладити пікове навантаження;

2) не тримати зайвих потужностей довше необхідного.

Це задача компромісу, в якій занадто раннє масштабування призведе до простою ресурсів і зайвих витрат, а занадто пізнє – до погіршення якості сервісу через перевантаження. Формально можна розглядати функцію витрат, що складається з двох компонент: вартість недовиділення ресурсів та вартість надмірного виділення. Оптимальний момент підключення нового ресурсу – коли сукупні маржинальні витрати від цих двох факторів мінімізуються [54].

Автоматичне масштабування ресурсів є ключовим механізмом управління високонавантаженими системами, що дозволяє підтримувати стабільну продуктивність при змінному робочому навантаженні, мінімізуючи витрати на обчислювальні ресурси. Його основна ідея полягає у динамічному коригуванні обсягу доступних ресурсів (процесорного часу, оперативної пам’яті, мережевих з’єднань) відповідно до фактичних та/або прогнозованих вимог системи.

Формально задача автоматичного масштабування має вигляд:

$$\min_{x(t)} \int_{t_0}^{t_f} \left[C\big(x(t)\big) + \lambda P(x(t), t) \right] dt, \tag{2.1}$$

де: C – функція вартості обчислювальних ресурсів; P – функція продуктивності системи; λ – ваговий коефіцієнт, що визначає пріоритет швидкодії

відносно витрат; t_0 – початковий момент часу; t_f – кінцевий момент часу, на якому завершується період оптимізації автоматичного масштабування; $x(t)$ – набір керованих параметрів системи, що визначає розподіл обчислювальних ресурсів у момент часу t, тобтонабір керованих під час масштабування з змінних, а саме:

– кількість активних віртуальних машин;

– кількість CPU, що використовується;

– обсяг оперативної пам’яті;

– кількість реплік;

– кількість мережевих потоків чи контейнерів.

Таким чином, зміна $x(t)$ впливає на продуктивність і витрати системи.

t_f може:

- визначати момент завершення прогнозованого піку навантаження, коли потрібно згорнути зайві ресурси (тобто кінець прогнозованого періоду навантаження), якщо система працює з проактивним масштабуванням;
- задавати часовий горизонт (тобто граничний момент часу аналізу), якщо рішення щодо масштабування приймається в рамках певного періоду (протягом доби або години);
- визначати фінальну точку у вікні оптимізації (тобто верхнею границею інтегралу, що обчислюється на ковзному інтервалі) у сценаріях, де використовуються методи динамічного керуванняє.

Вираз (2.1) не рівняння, а стандартна нотація в задачах оптимізації. Оператор $\min\limits_{x(t)}$ вказує, що відбувається пошук такого набору ресурсів (функції $x(t)$ у час t), для якого значення інтегрального функціоналу стає мінімальним. Іншими словами, справа від оператора min задано цільову функцію, яку потрібно мінімізувати, а ліва частина $\min\limits_{x(t)}$ визначає, що мінімізація проводиться по всіх можливих функціях $x(t)$.

(2.1) мінімізує сумарні витрати на ресурси $C(x(t))$ та штраф за низьку продуктивність $P(x(t), t)$ упродовж усього періоду від t_0 (початку управління масштабуванням) до t_f (кінця аналізованого відрізка часу).

Якщо $t_f \to \infty$, то маємо задачу довготривалої оптимізації, як у безперервно працюючій хмарній системі.

Якщо t_f обмежене коротким проміжком (порядку 10 хв), то оптимізація виконується лише в рамках цього періоду.

$P(x(t), t)$ – функція продуктвності системи, яка оцінює ефективність обробки запитів при даній конфігурації ресурсів $x(t)$ і може бути представлена як:

- середній час відповіді системи (час обробки одного запиту);
- частка запитів, що обробляються із затримкою вище допустимого порогу;
- сумарна кількість відхилених або неефективно оброблених запитів;
- λ – ваговий коефіцієнт, який визначає баланс між двома суперечливими критеріями:
- $C(x(t))$ – фінансові витрати на ресурси;
- $P(x(t), t)$ – швидкодія системи.

Якщо λ велике, система більше орієнтована на швидкодію (краще мати запас ресурсів). Якщо λ маленьке, система більше фокусується на мінімізації вартості, навіть якщо це призведе до деякого погіршення часу відповіді. Таким чином, модель дозволяє знаходити компроміс між продуктивністю та фінансовими витратами, вибираючи оптимальну кількість ресурсів $x(t)$ у кожен момент часу.

Масштабування у мікросервісних архітектурах та розподілених системах супроводжується додатковими викликами [55]:

- Неоднорідність навантаження між сервісами: деякі мікросервіси можуть масштабуватися автономно, тоді як інші залежать від загальної системної конфігурації;
- Горизонтальне масштабування або вертикальне масштабування: необхідність вибору між горизонтальним масштабуванням (збільшення кількості екземплярів сервісу) та вертикальним масштабуванням (виділення більш потужних ресурсів одному екземпляру);
- Міжсервісна взаємодія: під час зміни конфігурації одного сервісу може відбуватися каскадний ефект, що створює додаткові затримки у комунікації.

Для мінімізації негативного впливу необхідно впроваджувати координаційні механізми між мікросервісами, які дозволять синхронізувати процес масштабування і зменшити ризики перевантаження одного з компонентів системи.

Однією з основних проблем при реалізації автоматичного масштабування є необхідність відрізняти реальне зростання навантаження, що вимагає збільшення ресурсів, від аномальних стрибків, які можуть бути викликані одиничними подіями, помилками або атаками.

Помилкове реагування на такі аномалії може призвести до неоптимального масштабування, а саме:

- передчасного або надмірного розширення ресурсів, що спричиняє зайві фінансові витрати;
- ігнорування критичних змін у навантаженні, що може призвести до деградації продуктивності.

Автоматичне масштабування високонавантажених систем є необхідною умовою забезпечення їхньої продуктивності та економічної ефективності. Однак впровадження такої системи у практичне середовище супроводжується низкою викликів і обмежень, що стосуються як технічних, так і економічних аспектів. У цьому розділі розглянуто ключові проблеми, з якими стикаються розробники автоматизованих механізмів масштабування, а також фактори, що ускладнюють їхню адаптацію до реальних умов експлуатації.

Щоб мінімізувати вплив таких подій, застосовуються методи виявлення аномалій, які визначають, чи слід масштабувати систему або ігнорувати певні пікові значення [56].

У реальних умовах автоматичне масштабування має враховувати:

- Аномальні піки навантаження, що не відповідають історичним тенденціям;
- DDoS-атаки, які можуть імітувати органічне навантаження;
- Різкі коливання трафіку, спричинені змінами поведінки користувачів.

Наявність аномальних навантажень може призвести до неправильної реакції системи, що унеможливлює коректне масштабування.

Для вирішення цієї проблеми доцільно поєднувати такі класичні методи прогнозування з алгоритмами виявлення аномалій, як:

$$\text{IF } \lambda(t) > \mu + k\sigma, \text{ THEN ignore,} \tag{2.2}$$

де $\lambda(t)$ – коефіцієнт, що визначає чутливість моделі до аномалій; k – коефіцієнт, що визначає чутливість моделі до аномалій, μ – середнє історичне навантаження; σ – стандартне відхилення.

Таким чином, перевищення σ в 3 рази дозволить ігнорувати нетипові коливання, які не повинні впливати на процес прийняття рішень.

Вибір значення k є критичним:

- $k = 2$ означає, що відхилятиметься лише 5% найбільших сплесків у нормальному розподілі;
- $k = 3$ дає більшу стійкість до аномалій (~1% виняткових значень);
- $k = 5$ означає, що ігноруються лише екстремальні сплески, які трапляються рідко.

Якщо історичні дані показують, що середнє навантаження $\mu = 100$ запитів/с, а стандартне відхилення $\sigma = 20$. Якщо поточне навантаження $\lambda(t) = 160$, а використовується $k = 3$, то:

$$\mu + 3\sigma = 100 + 3 \times 20 = 160 \tag{2.3}$$

Отже, система вважає значення аномальним і не запускає процес масштабування. Це дозволяє підвищити точність автоматичного масштабування, мінімізуючи ризики помилкових рішень. Ця формула базується на статистичному критерії трьох сигм, який є частиною правила 68-95-99.7 (Empirical Rule) у нормальному розподілі [57].

Згідно формули 2.3:

- 68% значень знаходяться в межах $\mu \pm \sigma$;

- 95% значень – у межах $\mu \pm 2\sigma$;
- 99.7% значень – у межах $\mu \pm 3\sigma$.

Таким чином, якщо $k = 3$, то тільки 0.3% найвищих значень буде вважатися аномалією. Це ефективний спосіб визначення «нехарактерних» стрибків навантаження.

Автоматичне масштабування у високонавантажених системах часто використовує методи статистичного аналізу для ідентифікації аномальних значень у потоці запитів. Оскільки такі значення можуть бути спричинені помилками, збоєм або навмисними атаками (DDoS), важливо коректно відфільтровувати їх, щоб система не приймала неправильних рішень щодо масштабування.

Тому існує кілька варіантів розширення формули 2.3:

1. Використання ковзного середнього замість фіксованого середнього значення замість статичного μ використовують ковзне середнє (μ_{ma}), яке оновлюється в реальному часі:

$$\mu_{ma}(t) = \frac{1}{N}\sum_{i=t-N}^{t} \lambda(i), \tag{2.4}$$

де N – розмір вікна згладжування; а $\lambda(i)$ – фактичні значення навантаження за останні N періодів.

Формула (2.4) базується на методі ковзного середнього, що використовується у часових рядах для згладжування випадкових флуктуацій і трендового аналізу [58]. Головна перевага ковзного середнього полягає у тому, що воно дозволяє отримати гнучкішу оцінку середнього рівня навантаження, що адаптується до змін у реальному часі. N визначає «глибину» пам’яті моделі – чим більше N, тим менш чутливе середнє до короткострокових коливань. $\lambda(i)$ – значення навантаження, які використовуються для розрахунку середнього.

2. Виявлення аномалій на основі квантилів замість середньоквадратичного відхилення Якщо розподіл навантаження є негаусовим (містить довгі «хвости»),

стандартне відхилення може бути неефективним і в такому випадку можна використовувати методи інтерквантильного розмаху:

$$IF\ \lambda(t) > Q3 + 1.5 \times (Q3 - Q1),\ THEN\ ignore, \quad (2.5)$$

де $Q1$ і $Q3$ – це перший і третій квартиль відповідно.

В табл. 2.2 показано переваги та недоліки кожного з описаних методів.

Таблиця 2.2.

Порівняння підходів ідентифікації аномальних значень

Метод	Підхід	Переваги	Недоліки
$k\sigma$ (Стандартне відхилення)	Використання середнього і стандартного відхилення	Ефективний при нормальному розподілі даних	Нестійкий до асиметричних чи довгих «хвостів»
MA (Ковзне середнє)	Використання усереднення останніх значень	Адаптується до змін тренду	Чутливий до короткочасних сплесків
IQR (Інтерквантильний розмах)	Виявлення викидів через медіанні характеристики	Стійкий до негаусового розподілу	Потребує накопичення статистичних даних

Кожен з підходів має свої сильні та слабкі сторони, тому використання комбінованого підходу зазвичай дозволяє підвищити точність виявлення нетипових змін і уникати помилкових рішень при автоматичногму масштабуванні.

2.2 Реактивні методи масштабування систем

2.2.1 Метод порогових правил

Автоматичне масштабування у високонавантажених розподілених системах може бути реалізоване через реактивні чи проактивні методи керування ресурсами.

Реактивне масштабування базується на миттєвій реакції системи на зміни у завантаженості, тоді як проактивні методи використовують прогностичні моделі для завчасного коригування ресурсів (рис 2.1).

Рисунок 2.1. Класифікація методів автоматичного масштабування [59]

Обидва підходи мають переваги та обмеження, які залежать від точності прогнозів, швидкості реакції на зміну навантаження та фінансових витрат на утримання ресурсів.

Метод порогових правил один із найбільш поширених у високонавантажених системах і полягає в тому, що рішення про збільшення чи зменшення обчислювальних ресурсів приймається на основі встановлених порогових значень показників продуктивності, таких як використання CPU, оперативної пам’яті, тривалість обробки запитів або середня довжина черги повідомлень.

Реактивне масштабування на основі порогів передбачає безперервний моніторинг параметрів системи та прийняття рішень за таким алгоритмом:

– Визначається список ключових метрик, що відображають навантаження системи $M = \{m_1, m_2, \cdots, m_n\}$.

– Для кожної метрики встановлюється верхній та нижній пороги: m_i^{up} та m_i^{down}, де $i = 1, 2, \cdots, n$.

– Якщо значення метрики m_i перевищує верхній поріг m_i^{up}, система виконує масштабування вгору (додає ресурси).

– Якщо значення метрики опускається нижче нижнього порогу m_i^{down}, система виконує масштабування вниз (зменшує ресурси).

– Якщо значення метрики залишається у межах допустимого діапазону $m_i^{down} \leq m_i \leq m_i^{up}$, жодні дії не виконуються.

Метод порогових правил є простим в імплементації та широко використовується у промислових розгортаннях. Його основні переваги:

– Мінімальна обчислювальна складність: правила можуть бути реалізовані у вигляді простих умов без потреби в складних алгоритмах.

– Прозорість і передбачуваність: адміністратор має повний контроль над порогами та може легко відстежувати причини масштабування.

– Швидкість реакції: при правильному налаштуванні система може миттєво реагувати на пікові навантаження, уникаючи затримок.

Проте метод має і значні обмеження:

– Складність підбору оптимальних порогів: статичні значення можуть бути неефективними при змінних умовах навантаження.

– Чутливість до флуктуацій: якщо система зазнає короткочасних сплесків навантаження, можлива нестабільність через надто часті масштабування (ефект «гойдалки»).

– Нездатність до прогнозування: метод реагує лише на поточний стан системи, ігноруючи історичні дані та тренди.

В табл. 2.3 наведено порівняння ефективності методу порогових правил у різних сценаріях.

Як видно з таблиці, використання адаптивних порогів (через динамічний аналіз історичних значень метрик) може значно підвищити ефективність масштабування та зменшити частоту зайвих масштабувань. Таким чином, цей

метод залишається фундаментальним інструментом у реактивному масштабуванні, проте його ефективність значно зростає при використанні адаптивних механізмів налаштування порогів.

Таблиця 2.3.

Порівняння ефективності методу порогових правил у різних сценаріях

Метрика	Статичні пороги (без адаптації)	Адаптивні пороги (зі зміною порога залежно від трендів)
Стабільність (без гойдалок)	Середня	Висока
Час реакції на пікові навантаження	Швидкий	Швидкий
Кількість зайвих масштабувань	Висока	Низька
Витрати ресурсів	Потенційно вищі через зайві запуски	Оптимальні через точніші масштабування
Гнучкість для різних типів навантаження	Низька	Висока

Незважаючи на простоту, у нестабільних середовищах метод порогових правил може бути підданий ефекту нестабільного масштабування, тому в реальних застосуваннях часто комбінується з іншими методами, такими як методи теорії черг або машинного навчання [60].

Метод порогових правил найкраще підходить для систем, що характеризуються передбачуваним або поступово змінним навантаженням. Веб-застосунки, такі як онлайн-магазини, новинні портали та інформаційні сервіси, де сплески активності користувачів відбуваються у певні години дня, можуть ефективно використовувати цей підхід. У таких випадках порогові значення дозволяють швидко реагувати на зростання трафіку під час акційних розпродажів або випуску новин.

Попри свою гнучкість, метод не є ефективним для систем, що зазнають різких і нерегулярних змін навантаження. У випадках, коли сплески трафіку носять хаотичний характер або мають виражену сезонність, статичні порогові значення можуть виявитися неефективними, оскільки надто часто провокуватимуть масштабування або, навпаки, призводитимуть до дефіциту ресурсів. Системи з високими вимогами до SLA та низьколатентні сервіси, зокрема високочастотна біржова торгівля або обробка фінансових операцій, не можуть дозволити собі затримки, спричинені очікуванням перевищення порогового значення перед масштабуванням.

Недоцільним цей підхід є також у випадках, коли вартість розгортання нових ресурсів є суттєвою і кожне масштабування тягне за собою значні фінансові витрати. Якщо активація додаткових віртуальних машин або контейнерів супроводжується високими накладними витратами, система може не встигати масштабуватися достатньо швидко або робити це надто агресивно, породжуючи зайві витрати. У таких ситуаціях метод порогових правил поступається комбінованим або прогнозним підходам, які враховують історичні дані та тренди зміни навантаження [61].

2.2.2 Автоматичне масштабування на основі теорії черг

Автоматичне масштабування на основі теорії черг (Queueing Theory-Based Scaling, QTBS) є одним із ключових підходів до динамічного керування ресурсами у високонавантажених системах. Воно дозволяє балансувати між продуктивністю та вартістю, запобігаючи перевантаженню системи та забезпечуючи стабільний рівень обслуговування. Реалізація цього методу в Kubernetes-кластерах і мікросервісних архітектурах дає змогу автоматично додавати або зменшувати кількість реплік залежно від довжини черги запитів, що мінімізує час очікування й оптимізує витрати на інфраструктуру.

Цей метод спирається на математичне моделювання процесів обробки запитів, використовуючи класичні та розширені моделі теорії черг, що дозволяє

визначати необхідну кількість ресурсів залежно від інтенсивності вхідного потоку та часу обробки запитів.

Один із найпоширеніших підходів до моделювання масштабованої системи – використання моделі $M/M/s$, що описує багатоканальну систему з експоненційним розподілом інтервалів між надходженням запитів і часом обслуговування.

Алгоритм автоматичного масштабування з використанням теорії черг можна подати у вигляді таких кроків:

1. Моніторинг параметрів навантаження

– Періодично вимірюється λ, μ, поточний s, L_q та W_q.

2. Розрахунок ймовірності черги P_q

– Якщо $P_q > P_{max}$, ініціюється розширення системи.

3. Оцінка необхідної кількості серверів s'

– Виконується розрахунок нового значення s', що дозволить утримувати ймовірність черги нижче порогу.

4. Розгортання або згортання ресурсів

– Якщо ρ падає нижче певного мінімального значення (ρ_{min}), сервери можуть згортатися.

Коли веб-застосунок працює у середовищі Kubernetes і приймає запити через API Gateway, черга запитів може накопичуватися через змінне навантаження, і її довжина L_q використовується для прийняття рішення про масштабування.

У табл. 2.4 наведено приклад динамічного керування ресурсами залежно від довжини черги.

Як видно з таблиці, система нарощує кількість серверів у залежності від накопиченої черги, що дозволяє уникати перевантаження.

Автоматичне масштабування на основі теорії черг має значні переваги, серед яких адаптивність до реального навантаження, що дозволяє системі реагувати не лише на абсолютне зростання запитів, а й на параметри черги, такі як середній час очікування (W_q) та ймовірність відмови (P_q). Це важливо для мікросервісних архітектур, де обробка запитів розподіляється між кількома сервісами з різною

швидкістю обробки. Відмінністю цього підходу є те, що він дозволяє оптимізувати ресурси на основі внутрішніх параметрів, а не лише простих порогових значень використання CPU чи RAM, як у традиційних реактивних підходах.

Таблиця 2.4.

Динамічне масштабування на основі теорії черг

Довжина черги L_q	Поточна кількість серверів s	Визначена необхідна кількість s'	Дія
$L_q < 5$	2	2	Без змін
$5 \leq L_q < 10$	2	4	Додавання серверів
$10 \leq L_q < 20$	4	6	Додавання серверів
$L_q > 20$	6	10	Додавання серверів
$L_q < 3$	10	6	Видалення серверів

У системах обробки транзакцій у фінансовому секторі, де критично важлива мінімізація затримок і уникнення перевантажень, такий метод дозволяє підтримувати необхідний рівень продуктивності без зайвих витрат. Незважаючи на свої переваги, цей метод має певні обмеження. По-перше, він чутливий до точності оцінки параметрів: неправильний вибір порогового значення ймовірності зайнятості всіх серверів (P_{max}) або коефіцієнта використання ресурсів (ρ) може спричинити або надмірне масштабування, що збільшить витрати, або ж нестачу обчислювальних потужностей, що спричинить затримки. По-друге, у випадках різких піків навантаження система може не встигати масштабуватися через те, що теоретичні моделі черг, такі як $M/M/s$, передбачають відносно стабільний розподіл навантаження. Тому цей підхід менш ефективний у застосуваннях, де можливі раптові вибухові навантаження.

Оптимальним середовищем для застосування цього підходу є системи із передбачуваним і стабільним потоком запитів, такі як системи управління чергами

(RabbitMQ, Kafka), обробка запитів у мікросервісних API, а також платформи обробки замовлень і транзакцій у банківській сфері. Зокрема, у платіжних шлюзах, де навантаження змінюється протягом доби, але залишається стохастично передбачуваним, методи масштабування на основі теорії черг можуть забезпечити оптимальний баланс між продуктивністю та витратами. Натомість для систем із нестабільним або вибуховим навантаженням цей підхід може бути менш ефективним. У високодинамічних хмарних сервісах або потокових відеоплатформах, де навантаження змінюється миттєво, краще поєднувати теорію черг із проактивними підходами, які передбачають майбутнє навантаження на основі аналізу часових рядів або моделей машинного навчання [62].

Важливою сферою застосування є обчислювальні кластери даних для аналітики даних, де завантаження часто має довготривалий тренд із періодичними змінами – у таких системах алгоритми масштабування, що базуються на довжині черги і середньому часі очікування, дозволяють уникати перевантаження вузлів. Таким чином, автоматичне масштабування на основі теорії черг ефективне в системах із передбачуваними патернами навантаження але може бути менш результативним для систем із миттєвими сплесками трафіку, де необхідні комбіновані або проактивні методи масштабування [63].

2.2.3 Адаптивні реактивні методи

Реактивні методи автоматичного масштабування базуються на моніторингу показників продуктивності системи та прийнятті рішень щодо збільшення або зменшення ресурсів на основі заздалегідь визначених порогових значень. Однак класичні методи, такі як метод порогових правил, часто страждають від надмірної чутливості до короткочасних сплесків навантаження або, навпаки, не реагують достатньо швидко в умовах динамічно змінюваного середовища [64].

Саме тому адаптивні реактивні алгоритми, які модифікують порогові значення або параметри масштабування на основі історичних даних і поточного

стану системи, стали предметом активного дослідження в контексті підвищення ефективності автоматичного масштабування [65].

Одним із основних механізмів адаптивного реактивного масштабування є динамічне коригування порогів активації масштабування. Замість жорстко заданих порогових значень, таких як 80% використання CPU для додавання нових екземплярів сервісу, система адаптує ці пороги відповідно до поточного стану навантаження, середньої швидкості зміни метрики та історичних закономірностей.

Такий метод математично можна представити як динамічну зміну порога θ_t на основі попередніх значень метрики навантаження M_t:

$$\theta_t = \overline{M} + \lambda \cdot \sigma_M, \tag{2.6}$$

де: $\overline{\text{M}}$ – середнє значення метрики навантаження за певний проміжок часу, σ_M – стандартне відхилення навантаження, λ – коефіцієнт чутливості, що визначає адаптивність алгоритму.

У випадку високої мінливості навантаження поріг масштабування може бути знижений, щоб уникнути перевантаження до моменту запуску додаткових ресурсів, тоді як у стабільному режимі система може працювати на більш високих навантаженнях без передчасного розширення [66].

Замість запуску масштабування одразу при перевищенні порогу, реакція відбувається лише у випадку, якщо навантаження залишається в зоні критичних значень протягом певного інтервалу часу T_h, що можна виразити як:

$$S_{t+1} = \begin{cases} S_t + 1, & \text{M}_t > \theta_t \text{ протягом T}_h \text{ хв;} \\ S_t - 1, & \text{M}_t < \theta_d \text{ протягом T}_h \text{ хв;} \\ S_t, & \text{в інших випадках,} \end{cases} \tag{2.7}$$

де: S_t – поточний рівень масштабування, θ_t – поріг збільшення ресурсів, θ_d – поріг зменшення ресурсів, T_h – гістерезисний часовий інтервал.

Цей метод дозволяє запобігти коливанням кількості ресурсів при незначних змінних навантаженнях, особливо у високочутливих системах з великою кількістю запитів у секунду.

Найбільш ефективним рішенням для сучасних масштабованих систем є комбіноване використання реактивного масштабування з адаптивними алгоритмами, що дозволяє враховувати поточні особливості навантаження, не втрачаючи переваг швидкої реакції на пікові ситуації [67].

Адаптивні реактивні алгоритми значно покращують стабільність масштабованих систем, дозволяючи уникнути зайвих витрат та неконтрольованих змін конфігурації. Поєднання адаптивного методу з методами прогнозування дозволяє досягти оптимального балансу між швидкістю реакції та ефективністю використання ресурсів, що особливо важливо у високонавантажених розподілених середовищах [68].

2.3 Проактивні методи масштабування систем

2.3.1 Машинне навчання у проактивному масштабуванні

У сучасних високонавантажених системах автоматичне масштабування ресурсів є критичним для забезпечення стабільної роботи та ефективного використання інфраструктури. Існують два основні методи автоматичного масштабування: реактивний та проактивний. Кожен з них може бути реалізований на різних рівнях складності та автоматизації, умовно поділених на перший та другий рівні. Реактивне масштабування передбачає зміну кількості або потужності ресурсів у відповідь на поточні зміни навантаження.

Цей метод базується на моніторингу системних метрик, таких як використання CPU, пам'яті або мережевого трафіку, та прийнятті рішень про масштабування при досягненні певних порогових значень:

1. Реактивне масштабування першого рівня здійснюється за допомогою простих правил, де адміністративно встановлені пороги визначають, коли додавати

або видаляти ресурси. Якщо завантаження CPU перевищує 70%, додається додатковий екземпляр сервісу.

2. Реактивне масштабування другого рівня використовує більш складні алгоритми та моделі машинного навчання для аналізу історичних даних та поточних трендів. Це дозволяє передбачати короткострокові зміни навантаження та відповідно коригувати ресурси. Система може виявити, що певні події призводять до сплесків трафіку, і заздалегідь підготувати додаткові ресурси.

Проактивне масштабування спрямоване на передбачення майбутніх змін навантаження та завчасне налаштування ресурсів для їх обробки.

Цей метод базується на прогнозуванні та аналізі патернів використання системи:

1. Проактивне масштабування першого рівня використовує базові методи прогнозування, такі як аналіз часових рядів або сезонних коливань. Якщо відомо, що щоп'ятниці ввечері спостерігається підвищене навантаження, система автоматично збільшує ресурси в цей період.

2. Проактивне масштабування другого рівня застосовує складні моделі машинного навчання, включаючи глибокі нейронні мережі та алгоритми навчання з підкріпленням. Ці моделі аналізують великі обсяги даних, враховують різноманітні фактори та взаємозв'язки, що дозволяє точно прогнозувати навантаження та оптимально розподіляти ресурси. Система може враховувати не лише історичні дані, але й зовнішні фактори, такі як маркетингові кампанії або погодні умови, для більш точного прогнозування [69].

Схему роботи реактивного та проактивного машинного навчання показано на рис. 2.2.

Методи машинного навчання в автоматичному масштабуванні здебільшого працюють у двох режимах [70]:

1. Короткострокове прогнозування навантаження, де моделі аналізують історичні дані про трафік системи, виявляючи сезонні коливання, раптові сплески або нестандартні поведінкові патерни користувачів.

2. Автоматичне прийняття рішень щодо змін конфігурації ресурсів, де моделі ML використовують попередньо отримані прогнози для визначення оптимального рівня обчислювальних потужностей.

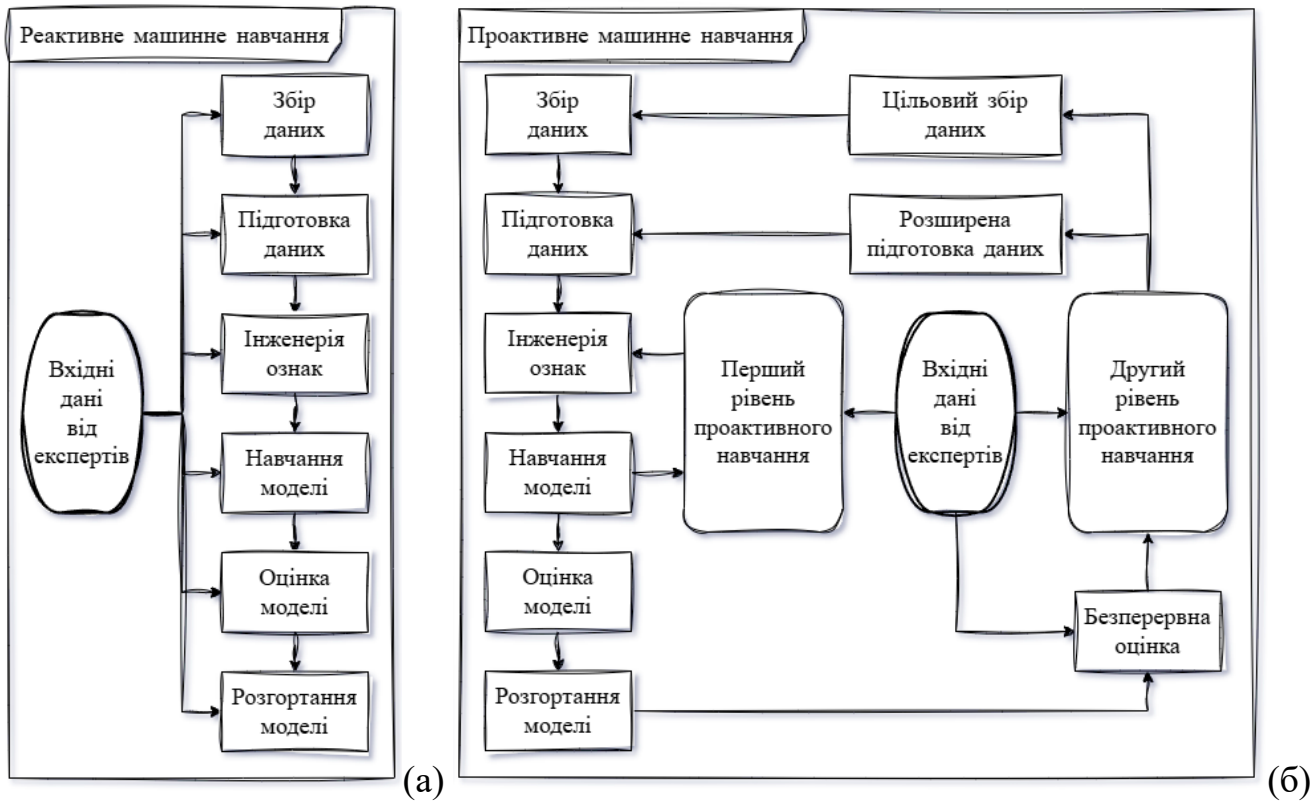

Рисунок 2.2. Схема роботи реактивного (а) та проактивного (б) машинного навчання [69]

Реалізація методу передбачає навчання моделі на великих наборах даних, що містять часові ряди використання ресурсів, дані про затримки обробки запитів, змінність навантаження у різні періоди доби та тижня, а також контекстні фактори, такі як активність користувачів [71]. Однією з ключових проблем, що виникають при застосуванні методів машинного навчання для прогнозного масштабування, є обмеженість точності передбачень, що може спричинити надмірне або недостатнє виділення ресурсів. Для зменшення цього ефекту використовують ансамблеві методи та гібридні підходи, що комбінують кілька моделей прогнозування.

Найчастіше у практиці автоматичного масштабування використовуються такі методи прогнозування:

1. Лінійні моделі (авторегресія (AR), авторегресійна модель із ковзним середнім (ARMA) та авторегресійна інтегрована модель (ARIMA)) демонструють добру ефективність за умови наявності чітко вираженої періодичності у даних [72]. Однак вони погано справляються з обробкою нерегулярних навантажень або неочікуваних змін у трафіку.

2. Глибокі нейронні мережі, зокрема рекурентні нейромережі (RNN) та їхні модифікації LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit), показують значно вищу точність у виявленні довгострокових залежностей та адаптації до змінного навантаження [73]. Вони ефективні для передбачення раптових змін трафіку, оскільки можуть враховувати довгу історію попередніх станів системи.

3. Моделі градієнтного підсилення (XGBoost, LightGBM, CatBoost) також ефективні у прогнозуванні короткострокових змін у навантаженні, особливо коли структура даних містить численні кореляційні ознаки [74].

4. Гібридні моделі, що поєднують статистичні методи (ARIMA) з нейромережевими підходами (LSTM, GRU), забезпечують найкращу узгодженість між передбачуваністю трендів і здатністю адаптуватися до непередбачуваних змін [75].

Застосування методів машинного навчання для прогнозування навантаження дозволяє зменшити ризик несподіваних перевантажень системи, а також знизити витрати на підтримку невикористаних обчислювальних потужностей. Найбільш ефективними для таких завдань є глибокі нейронні мережі (LSTM, GRU) та гібридні моделі (ARIMA+LSTM). У той же час використання ML-методів потребує додаткових обчислювальних ресурсів для тренування та оновлення моделей, що є важливим фактором при виборі кращого методу прогнозування.

2.3.2 Розширене навчання з підкріпленням

Навчання з підкріпленням (Reinforcement Learning, RL) – одним із перспективних методів реалізації проактивного масштабування у

високонавантажених розподілених системах. Використання RL дозволяє навчати агентів адаптивно приймати рішення щодо збільшення чи зменшення ресурсів на основі історичних даних, прогнозованого навантаження та поточного стану системи, мінімізуючи витрати та затримки в обробці запитів.

Один із найпоширеніших методів навчання агента в задачах автоматичного масштабування – Q-навчання (Q-learning), що використовує функцію цінності дій:

$$Q(s,a) = Q(s,a) + \alpha(R + \gamma \max_{a'} Q(s',a') - Q(s,a)), \quad (2.8)$$

де α – швидкість навчання.

Однак класичний Q-learning непридатний для реальних систем через високий розмір простору станів, що зумовило розвиток глибокого Q-навчання (Deep Q-Network, DQN), яке використовує згорткові нейронні мережі для апроксимації функції $Q(s,a)$.

Крім DQN, широко застосовуються методи політик, серед яких:

- Акторно-критичні алгоритми (A2C, A3C), що навчають нейромережеві агенти оптимізувати баланс між дослідженням нових стратегій і використанням уже знайдених;
- Proximal Policy Optimization (PPO), який обмежує розбіжність між новою та старою політикою агента, що запобігає нестабільності навчання;
- Deep Deterministic Policy Gradient (DDPG) для роботи з неперервними просторами дій, що особливо актуально для гнучкого масштабування ресурсів.

Окрім традиційних методів, гібридні методи поєднують методи навчання з підкріпленням із моделями прогнозування часових рядів, що дозволяє агенту не лише навчатися на історичних рішеннях, але й проактивно передбачати необхідне збільшення ресурсів.

У роботах [76, 77] доведено, що застосування DQN у поєднанні з LSTM-мережею для прогнозування навантаження дає змогу досягти зниження затримок обробки запитів на 17% при тому ж рівні використання обчислювальних ресурсів.

Навчання агента RL у хмарному середовищі відбувається у змінному просторі станів, що ускладнює процес оптимізації.

Ключовими факторами, які впливають на вибір методу навчання, є:

1. Гетерогенність середовища – різні типи ресурсів (CPU, RAM, IOPS) потребують різних стратегій масштабування.

2. Часова мінливість навантаження – агент повинен адаптувати свої рішення залежно від денного циклу роботи сервісів.

3. Швидкість навчання – необхідно балансувати між експлуатацією поточних знань і дослідженням нових рішень, особливо в середовищах з високими витратами на помилки.

Для прискорення навчання RL у хмарних системах часто використовуються імітаційні моделі, що дозволяють агенту віртуально випробовувати різні сценарії масштабування без впливу на реальні ресурси. Такий підхід дозволяє значно скоротити витрати часу на пошук оптимальних стратегій, особливо при використанні алгоритмів PPO та A3C, що ефективніше пристосовуються до непередбачуваних змін у середовищі [78].

Розширене навчання з підкріпленням демонструє високу ефективність у сценаріях динамічного управління ресурсами, що підтверджується численними експериментами. Застосування Deep RL для керування кількістю реплік у Kubernetes дозволяє зменшити середній час відповіді системи на 22% при одночасному скороченні витрат на 15% порівняно з традиційними реактивними методами [79].

Використання RL у масштабуванні може бути додатково оптимізоване за рахунок:

- Включення імітаційного навчання перед розгортанням агента у реальному середовищі.
- Використання мультиагентного RL, де кілька агентів приймають рішення одночасно для різних ресурсів.
- Адаптивного налаштування функції винагороди, щоб мінімізувати вплив короткострокових стрибків навантаження на вибір стратегії.

Таким чином, розширене навчання з підкріпленням є перспективним підходом до проактивного масштабування, що дозволяє динамічно адаптувати використання ресурсів відповідно до прогнозованого навантаження, мінімізуючи затримки та фінансові витрати. Водночас успішне впровадження таких методів потребує належного навчання агентів, балансування між стабільністю та адаптивністю стратегії, а також оптимізації параметрів середовища навчання.

2.3.3 Масштабування на основі нечіткої логіки

Автоматичне масштабування високонавантажених систем є завданням з високим рівнем невизначеності, оскільки динаміка навантаження рідко піддається точному прогнозуванню, а визначення жорстких порогових значень часто призводить до неефективного використання ресурсів. У таких випадках доцільним є застосування нечіткої логіки (Fuzzy Logic, FL), яка дозволяє обробляти невизначені або нечітко визначені вхідні параметри та приймати оптимальні рішення на основі гнучких правил.

На відміну від класичних методів, що покладаються на чіткі порогові значення, нечіткі системи оперують лінгвістичними змінними, такими як «низьке навантаження», «середнє завантаження» або «критичне завантаження», і приймають рішення на основі нечітких правил «якщо–то».

Це дозволяє адаптивно керувати масштабуванням навіть у випадках, коли немає чіткої межі між допустимим і недопустимим рівнем використання ресурсів. Нечітка логіка є розширенням класичної булевої логіки і оперує поняттями часткової істинності, що дозволяє моделювати людське міркування в умовах невизначеності. У контексті масштабування обчислювальних ресурсів нечітка логіка використовується для прийняття рішень щодо додавання або видалення ресурсів на основі нечітко визначених вхідних параметрів, таких як поточне навантаження системи, швидкість зміни навантаження та затримки у відповідях [80].

Процес прийняття рішень у системах на основі нечіткої логіки включає декілька етапів:

- Фазифікація: перетворення чітких вхідних даних у нечіткі змінні за допомогою функцій належності.
- Нечітке виведення: застосування набору правил типу "якщо-то", які визначають залежність між вхідними та вихідними змінними.
- Дефазифікація: перетворення отриманих нечітких висновків у чіткі значення, що визначають конкретні дії щодо масштабування.

Практичне впровадження масштабування на основі нечіткої логіки було досліджено в роботі [81], де автори запропонували систему горизонтального масштабування для Kubernetes-кластерів у контексті обчислень на периферії мережі. Запропонована система використовує нечіткий контролер для прийняття рішень про масштабування на основі таких параметрів, як середнє завантаження процесора та обсяг вхідного трафіку.

Нечіткий автоскейлер забезпечує баланс так: правила сформовано таким чином, щоб уникати як недостатнього виділення ресурсів, що призводить до деградації продуктивності, так і надлишкового виділення, що марно збільшує вартість.

Навіть якщо метрики CPU/пам’яті високі, але витрати вже близькі до максимальних, правила можуть рекомендувати лише помірне збільшення ресурсів або відкласти масштабування до критичного моменту. З іншого боку, коли навантаження спадає, нечітка логіка дозволяє плавно згорнути зайві ресурси, але не опускатися нижче мінімуму, необхідного для роботи.

Нечітка система фактично виконує багатокритеріальну оптимізацію в режимі реального часу, врівноважуючи ці два конкуруючі критерії в рамках заданих обмежень.

Результати експериментів показали, що використання нечіткої логіки дозволяє досягти більш стабільного та ефективного управління ресурсами. Зокрема, було відзначено зменшення середнього часу відгуку системи на 15% та зниження кількості непотрібних операцій масштабування на 20% у порівнянні з

традиційними методами, що базуються на жорстких порогових значеннях. Це свідчить про здатність нечітких систем адаптуватися до динамічних змін навантаження та приймати більш гнучкі рішення щодо розподілу ресурсів.

Крім того, впровадження нечіткої логіки сприяє оптимізації витрат на інфраструктуру. Завдяки точнішому налаштуванню кількості активних ресурсів відповідно до поточних потреб, вдалося знизити загальні витрати на обчислювальні ресурси приблизно на 12%. Це особливо важливо для компаній, що надають хмарні послуги, де ефективне використання ресурсів безпосередньо впливає на прибутковість.

Масштабування на основі нечіткої логіки є потужним інструментом для управління ресурсами у високонавантажених системах. Його здатність обробляти невизначеність та враховувати множинні фактори при прийнятті рішень дозволяє досягти кращої продуктивності та економічної ефективності. Практичні дослідження підтверджують, що впровадження нечітких контролерів у системи оркестрації контейнерів, такі як Kubernetes, сприяє зменшенню часу відгуку та оптимізації витрат на інфраструктуру, що робить цей підхід перспективним для широкого застосування в галузі хмарних обчислень та обчислень на периферії мережі.

Використання нечіткої логіки є особливо ефективним у контейнеризованих середовищах, таких як Kubernetes, де можливе поступове збільшення або зменшення кількості Pod, а також у функціональних хмарних обчисленнях (Serverless, FaaS), де швидке й адаптивне масштабування має критичне значення. Подальші дослідження у цій сфері можуть зосереджуватись на поєднанні нечітких систем із прогнозуванням часових рядів, що дозволить підвищити точність рішень щодо масштабування. Також перспективним є гібридний підхід, у якому нечітка логіка використовується разом із RL для навчання агента приймати оптимальні рішення на основі історичних даних. Таким чином, масштабування на основі нечіткої логіки демонструє високу адаптивність і ефективність у сценаріях з непередбачуваним навантаженням, зменшуючи латентність системи та підвищуючи її стійкість до короткочасних стрибків навантаження.

2.3.4 Масштабування на основі аналізу часових рядів

Підхід, що ґрунтується на аналізі часових рядів, дозволяє за допомогою статистичних методів прогнозувати майбутні коливання навантаження та ініціювати проактивне масштабування ресурсів задовго до досягнення критичних значень. Основною перевагою такого методу є можливість врахування сезонних і трендових компонентів у даних, що дозволяє своєчасно адаптувати обчислювальні потужності для забезпечення стабільної роботи системи.

Для прогнозування навантаження широко використовується метод експоненціального згладжування, який описується наступною формулою:

$$\hat{\lambda}_{t+1} = \alpha\lambda_t + (1-\alpha)\hat{\lambda}_t, \tag{2.9}$$

де λ_t – фактичне навантаження в момент часу t, а α є коефіцієнтом згладжування, що визначає вагу останніх спостережень. Також у випадках складних часових рядів застосовують ARIMA-моделі, які дозволяють враховувати сезонні коливання та трендові зміни, що особливо актуально для систем із високою динамікою роботи.

На рис. 2.3 зображено приклад аналізу часових рядів історичних даних навантаження на систему.

Сфера застосування даного методу охоплює автоматичне масштабування мікросервісів, управління контейнеризованими середовищами (Kubernetes) та системи обробки великих обсягів даних, де циклічність навантаження є типовою. Основною перевагою такого підходу є можливість завчасного планування розгортання додаткових ресурсів, що зменшує ризик перевантаження системи та сприяє зниженню часу реакції. Крім того, оптимізація економічних витрат досягається за рахунок адаптивного керування ресурсами – ресурси збільшуються лише у випадку реальної потреби.

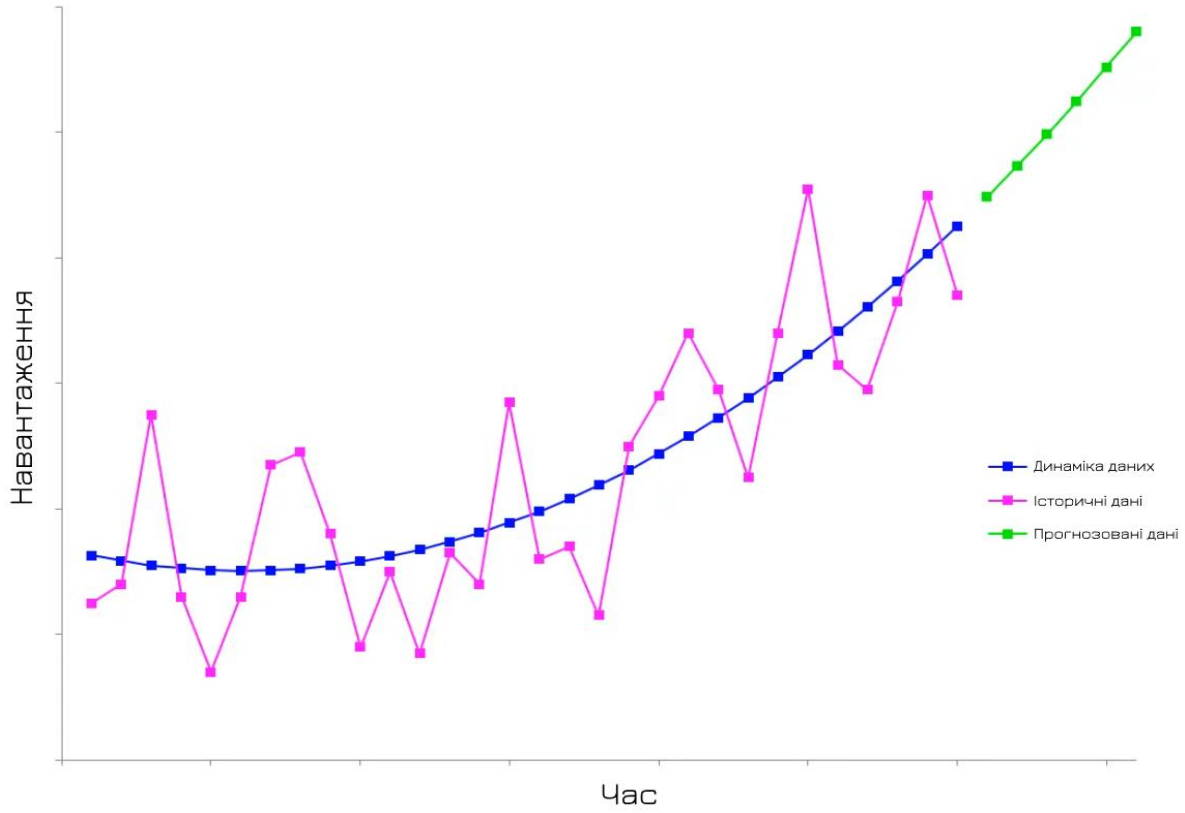

Рисунок 2.3. Динаміка навантаження системи

Проактивне масштабування, засноване на аналізі часових рядів, дозволяє визначити оптимальні часові інтервали для розгортання додаткових ресурсів.

Процес автоматичного масштабування на основі аналізу часових рядів можна представити як двоетапний механізм:

1. Прогнозування майбутнього навантаження $\hat{\lambda}(t+k)$ на основі історичних даних.

2. Прийняття рішення про масштабування, якщо прогнозоване навантаження перевищує певний поріг $\lambda_{\text{threshold}}$.

Якщо прогнозоване навантаження перевищує встановлений поріг $\lambda_{\text{threshold}}$, система заздалегідь генерує запит на розширення ресурсів, що дозволяє уникнути зниження якості обслуговування під час пікових періодів. Крім того, після завершення пікового навантаження за допомогою аналогічних методів визначається оптимальний момент повернення до базової конфігурації, що сприяє зниженню витрат на підтримку надмірних ресурсів. Серед основних переваг цього

підходу можна виділити адаптивність до циклічних змін, можливість мінімізації затримок за рахунок завчасного реагування, а також оптимізацію економічних витрат. Однак важливо зазначити, що точність прогнозування залежить від якості історичних даних і періодичного калібрування моделі, оскільки аномальні значення можуть призводити до помилкових прогнозів та невиправданого масштабування.

Основні підходи до прогнозування часових рядів для проактивного масштабування можна поділити на чотири класи:

1. Авторегресія (AR) описує значення часу через лінійну комбінацію попередніх значень із певним коефіцієнтом затухання. AR-моделі ефективні для стабільних трендових процесів без значних коливань та сезонності. Вони підходять для прогнозування навантаження в системах, де зміни відбуваються поступово. Проте вони погано працюють при наявності раптових піків або циклічних коливань, оскільки не враховують їх у своїй структурі. Їх доцільно використовувати коли навантаження змінюється поступово, без різких стрибків і має довготривалі тенденції. Але якщо в даних є короткочасні піки або регулярна сезонність, такий метод буде менш ефективним.

2. Модель ковзного середнього (MA) використовує середнє значення попередніх похибок для прогнозування майбутнього навантаження. На відміну від AR, метод MA ефективніше реагує на короткострокові коливання та випадкові зміни. Проте він не моделює довготривалі тенденції та не враховує періодичність. Цю модель доцільно використовувати коли система має випадкові короткі пікові навантаження. Та не підходить якщо потрібно прогнозувати довгострокові тренди або складні залежності.

3. Експоненціальне згладжування (ES) використовує вагове середнє всіх попередніх значень, де останні дані мають більшу вагу. Метод ES є найшвидшим у розрахунках і підходить для прогнозування короткострокових змін навантаження. Він добре підходить для високонавантажених мікросервісних архітектур, оскільки дозволяє швидко реагувати на зміни, не зберігаючи великі масиви історичних даних. Проте ES не здатне коректно прогнозувати сезонні та циклічні зміни. Цей

метод проявляє себе найкраще коли потрібно швидко адаптувати прогнозування до змін і немає складних залежностей. Але можуть виникати складнощі при довгостроковому прогнозі або періодичних коливаннях.

4. Авторегресійна інтегрована модель ковзного середнього (ARIMA) поєднує AR і MA, а також включає крок інтегрування для усунення нестаціонарності. Ця модель дозволяє прогнозувати нестаціонарні процеси, сезонні коливання та довготривалі тренди. Вона забезпечує високу точність для складних систем, проте має значну обчислювальну складність. Цю модель доцільно використовувати коли є сильна сезонність, циклічність або нестійкі тренди. Проте модель погано підходить для сценаріїв коли система вимагає миттєвих реакцій і швидких обчислень.

Порівняння даних методів, а також опис того як кожен з них справляється з різним типом проблем наведено у табл. 2.5.

Таблиця 2.5.

Порівняння методів аналізу часових рядів

Метод	Довготривалі тренди	Короткочасні піки	Циклічні процеси	Обчислювальна ефективність
AR	Добре прогнозує	Погано	Не враховує	Легка обробка
MA	Необхідне доповнення	Враховує	Не моделює	Легка обробка
ES	Обмежена точність	Реагує швидко	Не враховує	Дуже швидке
ARIMA	Найкраща точність	Враховує	Найкраще працює	Висока вартість

Для подальшого дослідження найкращим вибором є експоненційне згладжування, так як цей метод потребує мінімум витрат часу і процесорної потужності в умовах фінансових обмежень і мінімізації витрат. При цьому точність прогнозування сильно залежить від якості історичних даних та здатності моделі адаптуватися до раптових змін. Аномальні значення або короткострокові

коливання можуть спричиняти помилкові сигнали, що веде до непотрібного масштабування і, відповідно, до збільшення операційних витрат. Крім того, необхідність регулярного калібрування моделей для підтримки високої точності прогнозів може ускладнювати експлуатацію системи [82].

Метод масштабування на основі аналізу часових рядів знаходить своє застосування у багатьох сучасних хмарних системах, зокрема в середовищах управління контейнерами, таких як Kubernetes. Horizontal Pod Autoscaler (HPA) у Kubernetes може бути розширено шляхом інтеграції моделей прогнозування навантаження, що базуються на часових рядах, для ініціації проактивного масштабування заздалегідь до появи критичних пікових значень.

У сучасних дослідженнях запропоновано використання адаптивного прогнозування на основі експоненціального згладжування та ARIMA-моделей для прогнозування майбутніх змін навантаження. В [83] описано адаптивний алгоритм, який дозволяє точно прогнозувати пікові значення навантаження у хмарних системах, що призводить до зниження часу відповіді та оптимізації використання ресурсів у порівнянні з традиційними реактивними підходами. У роботі було продемонстровано, що застосування аналізу часових рядів для проактивного масштабування в Kubernetes-середовищах дозволяє досягти зниження середнього часу відповіді на рівні до 20% та значного зменшення експлуатаційних витрат.

Крім того, в [84] наведено порівняння гібридного підходу, який поєднує аналіз часових рядів із методами глибокого навчання, із традиційними реактивними методами масштабування. Результати дослідження свідчать, що комбінування статистичного прогнозування з моделями глибокого навчання забезпечує більш стабільне масштабування, зменшуючи ризики перевитрат через помилкове прогнозування та знижуючи затримки в обслуговуванні.

Важливо підкреслити, як ці методи прогнозування інтегруються в цикл автомасштабування. Як правило, окремий компонент (іноді його називають предиктором або службою прогнозування) працює безперервно або через певні проміжки часу. Він отримує останні дані моніторингу (кількість запитів в секунду

за останню годину) і видає прогноз на наступний інтервал або кілька наступних інтервалів.

Потім метод керований політикою автомасштабування інтерпретує цей прогноз. Якщо предиктор каже, що «через 5 хвилин навантаження буде вдвічі більшим», метод може вирішити подвоїти кількість екземплярів зараз. Деякі методи автоматичного масштабування використовують поріг для прогнозу – «якщо прогнозоване використання процесора через 5 хвилин перевищить 80%, тоді масштабуйте». Це об'єднує концепцію порогового значення з прогнозуванням. Інші використовують більш пряме відображення – «завжди тримайте прогнозований 95-й процентиль часу відгуку нижче SLA, виділяючи достатню кількість потужностей». Незалежно від специфіки, точність прогнозу безпосередньо впливає на ефективність масштабування.

Дослідники часто оцінюють моделі прогнозування з точки зору метрик стандартної помилки (MAE, RMSE) [85], але справжнім тестом є вплив на результати автомасштабування (дотримання SLA проти вартості). Деякі роботи моделюють автомасштабування з різними методами прогнозування, щоб побачити, який з них дає найкращий компроміс. Як правило, моделі часових рядів, які враховують сезонність та адаптуються до змін тренду (шляхом перенавчання або згладжування), мають тенденцію добре працювати в сценаріях автомасштабування, тоді як моделі, які не можуть пристосуватися до змін режиму або раптових подій, можуть призвести до втрачених можливостей масштабування.

Таким чином, масштабування на основі аналізу часових рядів демонструє високу ефективність у сценаріях, де навантаження має циклічний характер і може бути передбачене за допомогою сучасних моделей прогнозування. Застосування цього методу дозволяє завчасно планувати збільшення ресурсів, забезпечуючи високу якість обслуговування при оптимальних витратах. Проте його успіх значною мірою залежить від якості історичних даних та постійного калібрування моделей, що є ключовим чинником для впровадження у виробничі середовища [86].

2.4 Узгодження методів масштабування з архітектурними особливостями систем

Основним критерієм вибору між реактивним та проактивним масштабуванням є передбачуваність патернів навантаження. Якщо навантаження має впізнавані закономірності (щоденні цикли, тижневі тренди, періодичні сплески, пов’язані з відомими подіями), то проактивне масштабування є кращим вибором, оскільки воно дозволяє заздалегідь планувати коригування ресурсів відповідно до цих патернів. Онлайн-банківський мікросервіс може спостерігати передбачуваний сплеск кожного буднього дня під час обіду; проактивний масштабувач може навчитися цього патерну та виділити додаткові контейнери щодня о 11:50 перед початком зростання навантаження. Але якщо навантаження переважно непередбачуване (Випадкові сплески трафіку через вірусний контент або непередбачувану поведінку користувачів), необхідне реактивне масштабування, оскільки неможливо надійно прогнозувати такі події. Насправді багато систем демонструють комбінацію: базовий передбачуваний тренд плюс випадковий шум. Саме тому поєднання підходів часто є ефективним – проактивно обробляти базовий тренд, а реактивно – шум [87].

На вибір методу також суттєво впливає час, необхідний для того, щоб новий екземпляр мікросервісу став повністю готовим (затримка запуску), а також питання, чи є сервіс безстановим чи зі збереженням стану. Якщо мікросервіс може майже миттєво почати обслуговувати запити, реактивне масштабування може працювати навіть при помірно швидких змінах, оскільки затримка додавання ресурсів невелика. Проте, якщо сервіс має повільний запуск або ініціалізацію і завантажує велику модель машинного навчання в пам’ять або встановлює з’єднання, реактивне масштабування може бути занадто повільним – до того часу, як новий екземпляр стане готовим, сплеск навантаження може перевантажити систему. У таких випадках проактивне масштабування є більш вигідним, оскільки дозволяє приховати затримку запуску шляхом попереднього запуску екземплярів.

Аналогічно сервіси, що зберігають стан та підтримують дані сеансів користувачів або кешовану інформацію створюють додаткові виклики, їх масштабування може вимагати передачі стану або координації, що не може відбутися миттєво. Проактивний підхід може ініціювати реплікацію або розподіл стану заздалегідь, до збільшення навантаження. Сервіс кешування зі збереженням стану може проактивно реплікувати дані на новий вузол перед тим, як перенаправити частину навантаження.

Реактивне масштабування для таких компонентів часто призводить до затримки, коли новий екземпляр технічно працює, але ще не інтегрований (оскільки стан синхронізується), що негативно впливає на продуктивність. Таким чином, архітектурні аспекти, такі як витрати на запуск та накладні витрати на управління станом, спрямовують дизайн системи у бік проактивних методів для високонавантажених сервісів. Також мікросервіси рідко працюють ізольовано, вони викликають один одного для обробки запитів. У складній мікросервісній архітектурі реактивний підхід, який масштабує кожен сервіс окремо за власною метрикою, може призвести до помилок.

На рис. 2.4 зображена клієнт-серверна архітектура системи з двома сервісами.

Рисунок 2.4. Клієнт-серверна архітектура системи

Сервіс А (клієнт) викликає сервіс Б (сервер). З’являється хвиля навантаження на сервіс А. Реактивне масштабування сервісу А активується після збільшення використання оперативної пам’яті. Це спричиняє хвилю запитів до сервісу Б. До

того часу як балансувальник навантаження відреагує, сервіс А може вже перевантажити сервіс Б, що призведе до помилок або високої затримки.

Проактивний підхід передбачає, що при зростанні навантаження на сервіс А, сервіс Б також має масштабуватися одночасно (можливо, навіть трохи раніше, якщо масштабування сервісу Б відбувається повільніше). Для організацій, які можуть не мати достатньої експертизи, використання перевірених реактивних правил часто стає стандартним рішенням. Проте, із зростанням кількості готових інструментів прогнозуючого масштабування та успішними прикладами використання, впровадження таких методів зростає.

Операційні витрати на підтримку прогнозувальної моделі (розклад повторного навчання, перевірка її точності протягом часу) також є важливим фактором. Узгодження методів масштабування з архітектурою мікросервісів полягає у глибокому розумінні поведінки системи та її обмежень. Високонавантажені мікросервіси у складній архітектурній мережі отримують користь від проактивного, координованого масштабування, щоб уникнути каскадних перевантажень, тоді як незалежні безстанові сервіси можна ефективно масштабувати реактивно, керуючись окремими метриками.

Крім того, мікросервіси, що потребують більше часу для запуску або обробки стану, вимагають проактивних заходів для приховування цих затримок. Таким чином автоматичне масштабування не може бути універсальним рішенням – його необхідно адаптувати до архітектури додатку та профілю навантаження.

2.5 Розробка показників ефективності автоматичного масштабування

Оцінка ефективності автоматичного масштабування високонавантажених систем є критично важливим аспектом для вибору оптимальної стратегії керування ресурсами. Одним із ключових показників ефективності масштабування є середній час очікування запитів (T_w), що виникає у моменти недостатнього забезпечення ресурсами. В умовах реактивного масштабування цей параметр є критичним, оскільки система не має додаткових ресурсів до моменту перевищення порогових

значень. Для оцінки часу очікування використовується середній інтегральний показник, який визначається за формулою:

$$T_w = \frac{1}{N}\sum_{i=1}^{N} \max\left(0, T_i - T_{threshold}\right), \tag{2.10}$$

де N – загальна кількість запитів за певний період, T_i – час відповіді для i-го запиту, а $T_{threshold}$ – максимальний допустимий час відповіді, після якого система вважається перевантаженою. При використанні проактивного масштабування цей параметр має значно нижчі значення, оскільки ресурси вже підготовлені до моменту піку навантаження. Проте при надмірно ранньому масштабуванні можлива ситуація, коли розширені ресурси залишаються невикористаними, що призводить до фінансових втрат. Витрати на використання ресурсів (C_{total}) є другою важливою метрикою, яка безпосередньо впливає на вибір стратегії масштабування. Загальна сума витрат визначається як:

$$C_{total} = \sum_{j=1}^{M} C_j \cdot T_j, \tag{2.11}$$

де M – загальна кількість використаних конфігурацій ресурсів, C_j – вартість утримання ресурсів на рівні j (число активних віртуальних машин або контейнерів), а T_j – час, протягом якого система перебувала в цьому стані.

У реактивному масштабуванні значення C_{total} може бути нижчим через скорочений час використання пікової конфігурації, однак затримки у виконанні запитів можуть спричинити непрямі фінансові втрати, зокрема через SLA-порушення або погіршення користувацького досвіду. Проактивне масштабування, навпаки, передбачає вищі витрати через раннє розгортання ресурсів, але водночас забезпечує стабільну швидкодію системи. Проактивні стратегії масштабування базуються на прогнозах майбутнього навантаження, що можуть містити похибку (ΔP). Неточність прогнозу може призвести як до недостатнього масштабування, що спричиняє затримки та потребу в екстреному реактивному збільшенні ресурсів, так

і до надмірного масштабування, що призводить до зайвих фінансових витрат. Для оцінки похибки прогнозу можна використовувати середню абсолютну похибку (MAE):

$$\Delta P_{MAE} = \frac{1}{T}\sum_{t=1}^{T}\left|\lambda_{pred}(t) - \lambda_{actual}(t)\right|, \quad (2.12)$$

де $\lambda_{pred}(t)$ – прогнозоване навантаження в момент часу t, а $\lambda_{actual}(t)$ – реальне навантаження, зафіксоване системою моніторингу.

Чим вища неточність прогнозу, тим більше необхідно реактивних коригувань, що збільшує як середній час масштабування, так і витрати на екстрене розгортання ресурсів. Оптимальним рішенням є баланс між випереджувальним проактивним масштабуванням та гнучкими реактивними корекціями, що дозволяє компенсувати непередбачувані стрибки навантаження. Використання відповідних показників дозволяє забезпечити баланс між швидкодією, економічною доцільністю та стабільністю роботи системи. Основними критеріями, що визначають якість масштабування, є затримки у виконанні запитів, загальні витрати на інфраструктуру та частота коригувань конфігурації [88].

Оцінка впливу часу очікування на продуктивність системи є ключовою, оскільки невчасне масштабування може призвести до затримок у виконанні запитів та погіршення користувацького досвіду. Чим довше система перебуває у стані перевантаження без достатнього рівня ресурсів, тим більше часу очікування накопичується [89]. У реактивних методах масштабування цей показник часто є високим через необхідність очікування моменту, коли навантаження перевищить порогові значення. Водночас у проактивних методах можливе передчасне масштабування, що призводить до надмірного споживання ресурсів і підвищених витрат. Безперервний моніторинг затримок, витрат і точності прогнозів дозволяє адаптивно змінювати конфігурацію ресурсів залежно від умов експлуатації, що в довгостроковій перспективі зменшує фінансові витрати та підвищує загальну продуктивність системи.

2.6 Формалізація задачі оптимального масштабування

Автоматичне масштабування у високонавантажених системах є критичним компонентом для підтримки продуктивності та ефективного використання ресурсів. Традиційні підходи, що базуються на жорстких порогових значеннях, мають суттєві недоліки, оскільки не враховують динамічну природу навантаження та його варіативність у часі. Задання фіксованих граничних значень для CPU, пам’яті чи мережевої завантаженості може призводити як до надмірного масштабування та перевитрат, так і до недостатнього реагування у випадках раптового сплеску навантаження.

Порогове авто-масштабування страждає від статичності, а проблема фіксованих правил в тому, що система починає реагувати лише коли метрика вже вийшла за межі порогу, тобто коли проблема (перевантаження або недовантаження) вже проявилась. Це може призвести до запізнілого масштабування: різкий наплив користувачів за кілька хвилин перевищить поріг CPU, а додаткові VM запустяться із затримкою, коли частина запитів уже була оброблена повільно. Зворотна сторона – надлишкове масштабування: адміністратори часто змушені ставити консервативні низькі пороги, щоб гарантувати запас продуктивності, внаслідок чого ресурси додаються навіть при відносно невеликому рості навантаження, підвищуючи вартість. Правила з фіксованими порогами також погано поводяться при непостійних патернах. Якщо навантаження скаче, система може ввімкнутися-вимкнутися надто часто, або навпаки – не встигати повертатися у вихідний стан між сплесками.

Евристичні поліпшення, комбіновані умови або оптимізація порогів під історичні дані вирішують лише частину проблем. Вони, по суті, залишаються ручними налаштуваннями, що не гарантує оптимальності поза проєктними умовами. Зокрема, оптимальні параметри теоретичних моделей (параметри контролера чи довжини черги) можуть різко змінитися при змінах у профілі трафіку або конфігурації системи – те, чого аналітична модель не передбачала [90].

Методи на основі ML, здавалося б, адресують ці проблеми, але вводять свої виклики. Вони потребують історичних даних та навчання, тобто актуальні лише після того, як система вже відпрацювала достатній час для збору статистики. У випадку нових систем або нових типів навантаження ML-алгоритм може спершу працювати неточно. Навіть після навчання, модель має схильність до помилок, якщо реальні умови відрізняються від тих, що були в тренувальних даних. Разовий сплеск, спричинений непередбачуваною подією не буде належно передбачений, бо не вписується в історичний шаблон. Система тоді або не масштабуватиметься завчасно, або, якщо включити великий запас, буде тримати зайві ресурси більшу частину часу. Більше того, ML-моделі оптимізуються під певну метрику (часто – CPU), і не завжди враховують багатофакторність ухвалення рішення. Модель може прогнозувати лише навантаження CPU і масштабувати по ньому, тоді як справжня продуктивність сервісу може залежати ще й від затримок БД, об'єму пам'яті або зовнішніх подій.

Налаштування моделі під всі важливі фактори – складна багатовимірна задача, яку складно розв'язати універсально. Крім того, як згадувалось, чисто ML-рішення менш інтерпретовані і прозорі. У критичних системах (фінансових, медичних) це може бути неприпустимим з точки зору аудиту та довіри [91]. Нечітка логіка сама по собі вже додає гнучкості у процес масштабування – на відміну від порогового методу, вона може плавно адаптувати рішення під поточні умови. Це зменшує ризик різких стрибків і коливань кількості ресурсів, допомагає підтримувати стабільнішу продуктивність. Це і є приклад балансу між продуктивністю та вартістю: система не тримала зайвих VM без потреби, але й не допускала значних просідань швидкодії. Інша перевага – нечітка логіка легко комбінує декілька вхідних метрик. Можна одночасно врахувати CPU, пам'ять, і довжину черги запитів, задавши правила, що відображають цілісну картину навантаження. Це допомагає уникнути ситуації, коли масштабування спрацьовує по одному параметру, але є неадекватним з точки зору іншого. Чисто CPU-орієнтований скейлер може не реагувати на зростання часу відповіді, якщо CPU ще не досяг порогу – у нечіткої системи таке можна врахувати.

Головне, що дає нечітка логіка – стійкість до невизначеності та шумів у даних. Порогові методи можуть реагувати на сплески хибно. Нечіткий же контролер сприймає CPU = 75% не як абсолютно критичне значення, а як певну ступінь належності до множини High Load, можливо з урахуванням контексту інших метрик. Рішення (наскільки збільшити ресурси) буде більш пропорційним реальній потребі, а не бінарним. В результаті система краще витримує випадкові коливання без зайвих перекидок. Водночас, щоб повністю розкрити потенціал нечіткого підходу, потрібна його адаптивність. Саме тому пропонується розробити нову модель – адаптивну нечітку модель – яка б навчалась та підлаштовувалась по мірі зміни умов.

Існуючі реалізації нечітких авто-скейлерів починають рухатися в цьому напрямку. Адаптивна нечітка модель могла б поступово зміщувати порогові області термів (низький, середній, високий) в залежності від того, чи часто вона виявляється надто консервативною чи агресивною. Або ж коригувати вагу певних правил: якщо правило спрацювало і призвело до перевищення затримки, система зменшить силу масштабування за таких умов наступного разу.

Таким чином, відбувається самоналаштування авто-скейлера під конкретну систему й динаміку її навантаження. Це особливо цінно в умовах, коли робочі навантаження еволюціонують – веб-сервіс набуває нових користувацьких патернів, або додаються нові сервіси, що змінюють профіль споживання ресурсів. Адаптивна нечітка модель зможе вчитися на таких змінах, тоді як статичні порогові чи нечіткі правила потребували б ручного тюнінгу, а класичний ML-модуль вимагав би перевчення на нових даних [92]. Варто підкреслити, що необхідність подібних адаптивних рішень відзначають і інші дослідники. В огляді [93] методів авто-масштабування на базі нечіткої логіки (2024) зазначено, що хоча ML та глибокі нейронні мережі широко досліджені, уваги потребує адаптивність і інтерпретованість рішень на основі нечіткої логіки. Нечіткі контролери здатні ефективно працювати в умовах варіативності попиту, але нині бракує підходів, які поєднують цю гнучкість із здатністю автоматично підлаштовуватися під нові патерни.

Висновки до розділу 2

1. Показано необхідність адаптивного управління ресурсами в умовах аномальних та непередбачуваних навантажень. Розглянуто вплив динамічних змін у запитах на ефективність масштабування та виявлено, що жорсткі порогові значення можуть призводити до неефективного розподілу ресурсів, що підтверджує доцільність використання більш гнучких стратегій масштабування.

2. Встановлено, що реактивні методи автоматичного масштабування, серед яких метод порогових правил, автоматичне масштабування на основі теорії черг та адаптивні реактивні методи, є ефективними у випадках різких змін навантаження, однак вони мають низку обмежень, що пов’язані з запізненням у реакції на зміну системних параметрів. Обґрунтовано необхідність комбінування реактивних методів із проактивними підходами для підвищення ефективності управління ресурсами.

3. Зясовано, що використання проактивних методів автоматичного масштабування, зокрема машинного навчання, розширеного навчання з підкріпленням, нечіткої логіки і аналізу часових рядів, дозволяє передбачати навантаження і завчасно ініціювати масштабування, що зменшує ризик перевантаження системи. Обґрунтовано необхідність розробки комбінованих стратегій, які дозволять поєднувати прогнозні підходи з механізмами негайного реагування на зміну навантаження.

Основні наукові результати по цьому розділу опубліковані в працях [17].

РОЗДІЛ 3. АДАПТИВНА НЕЧІТКА РЕГРЕСИВНА МОДЕЛЬ ДИНАМІЧНОГО МАСШТАБУВАННЯ РЕСУРСІВ

У розділі запропоновано адаптивну нечітку регресивну модель динамічного масштабування ресурсів; на основі цієї моделі розроблено адаптивний нечіткий регресивний метод (АНРМ), який забезпечує гнучке керування обчислювальними ресурсами високонавантажених розподілених систем; обґрунтовано доцільність застосування нечіткої логіки для оцінювання поточного стану навантаження, що дає змогу уникнути жорстких порогових значень і здійснювати плавне масштабування; у межах АНРМ розроблено систему нечіткого виведення й сформовано матрицю ваг термів; проведено дослідження впливу характеристик хвилі навантаження на ефективність масштабування і показано, що невдалий вибір моменту зміни конфігурації може спричинити перевитрати або зниження продуктивності; розроблено метод коригування моменту масштабування, який аналізує вихідні величини АНРМ і визначає найкращий час зміни конфігурації, запобігаючи частому перемиканню станів системи; описано принцип використання розрахованого оптимального часу згортання ресурсів; показано, що застосування запропонованого методу, який інтегрує експоненційне згладжування з подвійною корекцією й нечітку регресивну оцінку, усуває надмірну реактивність класичних методів та забезпечує стабільну динаміку адаптації ресурсів до змін навантаження.

3.1. Концепція нечіткої моделі автоматичного масштабування

3.1.1 Постановка задачі масштабування та ключові критерії

Задача масштабування ресурсів у високонавантажених системах традиційно формулюється з огляду на дві суперечливі мети [93]:

1) забезпечення належного рівня продуктивності;
2) мінімізація витрат на утримання обчислювальної інфраструктури.

Перша мета вимагає від системи вчасного розширення ресурсів за зростання завантаження, оскільки дефіцит процесорних ядер чи пам’яті призводить до порушень угоди про рівень сервісу, збільшення часу відгуку, а в гіршому випадку – до відмов у обслуговуванні.

Друга мета, навпаки, спонукає не підтримувати надмірного запасу потужностей, оскільки навіть за короткочасної недозавантаженості марно витрачаються кошти.

У типовому випадку користувацькі навантаження характеризуються значними добовими або тижневими флуктуаціями, а також епізодичними різкими сплесками, викликаними зовнішніми подіями, рекламними кампаніями чи сезонною активністю. Статичні порогові методи масштабування недостатньо точно прогнозують такі піки та спади.

Дотримання лише одного порога для центрального процесора, може призвести до ігнорування проблем дефіциту оперативної пам’яті або навпаки – необґрунтованих витрат при одночасно низькому завантаженні більшості компонентів [94].

У контексті багатокритеріальної постановки задача масштабування інтерпретується так, що для кожного моменту часу слід визначити, які саме ресурси ядра процесора та гігабайти оперативної пам’яті) потрібні, аби гарантувати пропускну спроможність і мінімізувати сукупні витрати. Нечітке агрегування цих оцінок у процесі прийняття рішення дає можливість побудувати універсальні правила, які враховують складність реальних патернів навантаження. Така модель, на відміну від методів жорстких порогів, зберігає здатність підлаштовуватися під ситуації, що не були явно передбачені на етапі проєктування. Особливої актуальності це набуває для розподілених мікросервісних архітектур, у яких різні компоненти системи можуть мати суттєво відмінні профілі використання ресурсів [95].

Зважаючи на потребу проактивно визначати моменти майбутнього перевищення допустимих рівнів навантаження, постає додаткова задача – прогнозувати динаміку метрик у коротко- та середньостроковому періодах. Якщо

використовувати тільки реактивну логіку, система вчасно не запустить додаткові вузли, оскільки ключові показники (CPU) зростуть до критичних порогів раптово.

Саме тому адаптивна нечітка модель, поєднана з механізмом експоненційного згладжування часових рядів, здатна вчасно розпізнавати тренд чи періодичність навантаження і реагувати до настання перевантажень [96].

Таким чином, постановка задачі масштабування полягає у виборі оптимального обсягу ресурсів на кожен момент часу з урахуванням багатьох факторів та чітко вираженої невизначеності. Ключовими критеріями залишаються уникнення деградації продуктивності (шляхом збільшення обчислювальних і пам’ятних ресурсів) та економічна доцільність (запобігання нераціональному перезакладенню потужностей) [97].

3.1.2 Нечітке представлення даних про навантаження

Для створення адаптивної моделі першочергово необхідно формалізувати поняття «навантаження» та пов’язати його з потенційними діями масштабування. Завантаженість центрального процесора і використання оперативної пам’яті набувають значень у широких діапазонах, коливання в межах яких не завжди мають однозначну інтерпретацію у вигляді «високе» чи «низьке» навантаження [98].

Так, ситуація, коли сумарне завантаження CPU становить приблизно 50%, може вважатися ще далеко не критичною або, навпаки, потенційно небезпечною залежно від інших умов. Класичні порогові методи розглядають 50% як певну жорстку межу, котру система інколи перетинає навіть за короткотермінових сплесків, створюючи помилкові сигнали для розгортання додаткових потужностей [99].

У нечіткій моделі пропонується визначити для кожної метрики кілька лінгвістичних рівнів, позначаючи їх термами «Low», «Medium», «High» та «Very High». При цьому кожний із цих термів описується функціями належності, що поступово зменшують свій ступінь від 1 до 0, охоплюючи перехідні зони між

суміжними станами. Нечіткі множини для завантаження CPU та пам’яті можуть бути реалізовані через трикутні або трапецієподібні функції, оскільки вони спрощують обчислення та зручні для налаштування [100].

Адаптивність повинна ще посилити цей ефект, оскільки система зможе тонше підлаштовуватися під патерни запитів. Крім того, очікується зменшення коливань кількості інстансів – адаптивна нечітка модель зможе вчитись уникати зайвого масштабування, якщо побачить, що певні сплески короткочасні і краще їх перекрити запасом продуктивності на існуючих вузлах. Це підвищить стабільність конфігурації і спростить управління (менше операцій запуску/зупинки VM означає менше накладних витрат і потенційних помилок) [102].

Одночасно високі навантаження на CPU і пам’ять у поєднанні з обмеженнями бюджету вимагають більш гнучкого підходу, ніж проста перевірка порогу. Нечітка логіка пропонує формальний апарат для врахування невизначеностей і одночасного врахування кількох метрік при ухваленні рішень щодо масштабування [103].

Реалізація нечіткого представлення даних дає змогу ефективно працювати з шумовими чи флуктуальними сигналами. Якщо проміжна «середня» зона належності сформована достатньо широко, то короткочасні «піки» не призводять до надмірної реакції, а натомість вказують на часткову активацію правил, зорієнтованих на потенційний, але ще не критичний стан. Такий підхід уможливлює витриманий перерозподіл ресурсів, залежно від швидкості зростання або зниження показників у часі [104].

Крім того, функції належності можуть бути модифіковані відповідно до поточних або нових патернів навантаження. Якщо історичний аналіз показує, що система часто перебуває у близько 70% завантаження CPU, такий стан можна врахувати окремим рівнем або змістити межі між «середнім» і «високим» рівнями задля уникнення надто частих дій масштабування. Саме такий механізм самоналаштування є визначальним у побудові адаптивної нечіткої моделі динамічного масштабування ресурсів [105].

3.2. Формування математичної моделі

3.2.1 Визначення вхідних і вихідних параметрів

Структуризація вхідних даних у нечіткій системі завжди починається з ідентифікації тих параметрів, що найістотніше впливають на рішення про динамічне масштабування. У контексті високонавантажених розподілених систем первинне значення має завантаження процесорних ядер та ступінь використання оперативної пам’яті, оскільки саме їх нестача першочергово викликає зростання часу відгуку й ризик збоїв [106].

Вхідні дані являють собою множину значень:

$$\mathrm{X} = \{(L_{CPU}, C), (L_{RAM}, R)\} + (Y_{cpu}, Y_{ram}), \tag{3.1}$$

де:

$L_{CPU} = \{l_1, l_2, \dots, l_N\}$ – множина виміряних значень завантаження активних процесорів (у відсотках, тобто, $l_i \in [0{,}100]$ для $i = 1, \dots, N$;

C – системний ліміт для CPU (максимальна кількість доступних процесорів);

L_{RAM} – поточний обсяг використовуваної оперативної пам’яті;

R – системний ліміт для RAM (максимальний доступний обсяг пам’яті);

$Y_{cpu},\ Y_{ram}$ – не подаються на вхід явно, але використовуються фоново. Це попередні прогнозовані значення отримані за допомогою експоненційного згладжування з корекцією та подальшої нечіткої апроксимації (необов’язкові, використовуються для покращення точності прогнозування).

Для CPU вхідні дані представляють собою множину відсоткових значень завантаження кожного з процесорів: $L_{cpu} = \{l_1, l_2, \dots, l_N\}$ – це значення завантаження окремих процесорів від 0 до 100%, N – фактична кількість процесорів, що використовуються, при цьому N може бути меншим або рівним системному ліміту C (максимальна кількість доступних процесорів).

Для оперативної пам’яті (RAM) на вхід подається поточний обсяг використовуваної пам’яті L_{ram} (у гігабайтах) та системний ліміт R (максимальний доступний обсяг пам’яті). Нормування дозволяє перевести абсолютні показники в універсальний масштаб [0; 1], що є критично важливим для забезпечення адаптивності моделі незалежно від апаратних характеристик системи [107].

Нормування здійснюється за наступними формулами:

$$\lambda_{cpu} = \frac{1}{C \times 100} \sum_{i=1}^{N} l_i \qquad \lambda_{ram} = \frac{L_{ram}}{R} \tag{3.2}$$

В табл. 3.2 наведено формалізовану схему позначень для моделі, що розробляється.

Таблиця 3.2

Формалізована схема позначень для CPU та RAM

Позначення для CPU	Позначення для RAM	Опис	Діапазон
L_{cpu}	L_{ram}	Абсолютні виміряні значення завантаження: для CPU – множина завантаження процесорів (у відсотках), для RAM – обсяг пам’яті у ГБ	[1, C/R]
C	R	Системні ліміти ресурсів: максимальна кількість доступних процесорів та максимальний обсяг пам’яті	$\mathbb{N}^+$
λ_{cpu}	λ_{ram}	Нормовані поточні значення навантаження (використовуються для нечіткої інтерпретації)	[0, 1]
$\hat{\lambda}_{cpu}$	$\hat{\lambda}_{ram}$	Нормовані прогнозовані значення отримані методом експоненційного згладжування	[0, 1]
$\tilde{L}_{cpu}$	$\tilde{L}_{ram}$	Термічні (лінгвістичні) оцінки прогнозованих або фактичних значень	–
Y_{cpu}	Y_{ram}	Нормовані керуючі показники для масштабування отримані шляхом нечіткої апроксимації	[0, 1]
N_{cpu}	N_{ram}	Остаточні обчислені ненормовані значення як рекомендації для майбутнього масштабування обмежені лімітом системи	[1, C/R]

Вихідні дані – множина прогнозованих значень майбутнього масштабування (явний результат роботи методу), а також фактичні та прогнозовані нормовані значення (збережені для подальшого використання при наступному застосуванні):

$$Y = \{(N_{cpu}, N_{ram})\} + (\lambda_{cpu}, \lambda_{ram}), (Y_{cpu}, Y_{ram}) \quad (3.3)$$

Метод рекомендується використовувати через рівні проміжки часу для досягнення найкращого результату роботи. Чим менший проміжок – тим краще метод буде прогнозувати необхідні для системи ресурси.

3.2.2 Інтеграція експоненційного згладжування

AHPM поєднує механізми часового прогнозування, нечіткої логіки та експоненційного згладжування для досягнення оптимального балансу між продуктивністю та вартістю. Головною проблемою традиційних методів є їхня жорсткість – вони або реагують із запізненням, що викликає короткочасні провали в обслуговуванні, або створюють надлишкове масштабування, що призводить до зайвих витрат. Використання експоненційного згладжування дозволяє гнучко передбачати моменти пікового навантаження та ефективно адаптувати ресурси [108].

Класичне експоненційне згладжування використовує формулу [109]:

$$X_{next} = \alpha \cdot X_{current} + (1 - \alpha) \cdot X_{previous}, \quad (3.4)$$

де α – коефіцієнт згладжування, який контролює, наскільки сильно AHPM орієнтується на останнє спостереження. Якщо α близьке до 1, прогноз більше залежить від поточного значення, тоді як при нижчих значеннях відбувається врахування попередніх тенденцій. Це дозволяє уникнути надмірної чутливості до випадкових коливань навантаження і оперативно реагувати на системні зміни.

Однак у AHPM використовується модифіковане експоненційне згладжування з подвійною корекцією (похибки прогнозу та тренду):

$$\hat{\lambda}_{cpu}^{next} = \alpha \cdot \lambda_{cpu}^{now} + (1-\alpha) \cdot \hat{\lambda}_{cpu}^{prev} + \beta \cdot \left(\lambda_{cpu}^{now} - \hat{\lambda}_{cpu}^{prev}\right) + \gamma \cdot \left(\lambda_{cpu}^{now} - \lambda_{cpu}^{prev}\right), \quad (3.5)$$

$$\hat{\lambda}_{ram}^{next} = \alpha \cdot \lambda_{ram}^{now} + (1-\alpha) \cdot \hat{\lambda}_{ram}^{prev} + \beta \cdot \left(\lambda_{ram}^{now} - \hat{\lambda}_{ram}^{prev}\right) + \gamma \cdot \left(\lambda_{ram}^{now} - \lambda_{ram}^{prev}\right), \quad (3.6)$$

де: λ_{cpu}^{now}, λ_{ram}^{now} – поточні фактичні значення навантаження; λ_{cpu}^{now}, λ_{ram}^{now} – попередні фактичні значення навантаження (отримані $\lambda_{cpu}, \lambda_{ram}$ в результаті попередньої роботи); $\hat{\lambda}_{cpu}^{prev}$, $\hat{\lambda}_{ram}^{prev}$ – попередні прогнозовані значення на поточний момент (отримані Y_{cpu}, Y_{ram} в результаті попередньої роботи); $\hat{\lambda}_{cpu}^{next}$, $\hat{\lambda}_{ram}^{next}$ – оновлені прогнози на майбутнє; $\alpha, \beta, \gamma \in [0,1]$ – коефіцієнти впливу: α – основний коефіцієнт експоненційного згладжування, який визначає баланс між довірою до нового виміряного значення та інерційністю прогнозу; β – коефіцієнт корекції похибки попереднього прогнозу. Застосовується для швидкого усунення систематичної помилки в прогнозах; γ – коефіцієнт короткочасної тенденції, що враховує напрям і швидкість зміни навантаження.

У AHPM коефіцієнти α, β та γ задаються вручну, залежно від характеристик конкретної системи, вимог до точності прогнозу та чутливості до змін у навантаженні.

Ці коефіцієнти не залежать від статистичного навчання чи оптимізації, що робить метод простим для реалізації, прозорим для інтерпретації та адаптивним до практичних умов.

α – основний параметр експоненційного згладжування. Визначає вагу, яка надається останньому фактичному значенню навантаження.

Велике α (0.7–0.9) – метод швидше реагує на поточні зміни, менш інерційний.

Мале α (0.3–0.5) – метод більш згладжений, менш чутливий до короткочасних стрибків.

β – коефіцієнт корекції похибки попереднього прогнозу. Показує, наскільки важливо враховувати помилку, допущену раніше.

Середні значення β (0.1–0.3) дозволяють згладжено виправляти помилки.

Високе β (0.4–0.6) краще підходить у нестабільних системах, де точність прогнозу змінюється динамічно.

Мале β або нуль – ігнорування минулої похибки.

γ – коефіцієнт корекції за трендом. Враховує темп зростання або падіння навантаження між двома останніми точками. Це важливо, коли навантаження змінюється монотонно.

Середні значення γ (0.1–0.3) забезпечують реакцію на тренд без перенасичення.

Високе γ (>0.4) застосовують, якщо система має інерційні піки.

Таблиця 3.3 описує рекомендації щодо вибору значень коефіцієнтів для різних типів систем.

Таблиця 3.3

Рекомендації щодо вибору значень

Характеристика системи	α	β	γ	Коментар
Система чутлива до перевантажень, потрібно швидко реагувати	0.8–0.9	0.2–0.3	0.2–0.3	Пріоритет швидкої реакції, навіть із ризиком помилкового масштабування.
Система із коливаннями, важлива стабільність	0.4–0.6	0.2–0.4	0.1–0.2	Уникаємо переналаштувань при кожному сплеску, компенсуємо неточність.
Система з прогнозованими піками	0.6–0.7	0.3–0.4	0.3–0.4	Збалансований підхід: враховуємо похибку та тренд при стабільному патерні.
Система нечутлива до коротких сплесків, важлива економія	0.3–0.5	0.1–0.2	0.1	Мінімізуємо масштабування через короткі піки, система працює інерційно.

Коефіцієнти α, β та γ є настроюваними параметрами, які визначають поведінку механізму прогнозування. Їх доцільно обирати емпірично з урахуванням

особливостей цільового середовища. У системах з динамічною зміною навантаження важливо забезпечити компроміс між чутливістю до піків і стійкістю до шуму та компенсувати похибки попередніх прогнозів без перерегулювання [109]. Задання цих параметрів вручну дозволяє гнучко адаптувати метод до конкретних сценаріїв експлуатації.

3.2.3 Нечітке представлення даних про навантаження

На основі отриманих прогнозних значень навантаження виконується їхня фазифікація. Кожне з передбачених значень $\hat{\lambda}_{cpu}$ та $\hat{\lambda}_{ram}$ оцінюється щодо належності до певних нечітких множин. Це дозволяє визначити, наскільки значення знаходиться між двома категоріями, і уникнути ситуацій, коли навіть незначна зміна завантаженості спричиняє різку зміну масштабування.

Щоб відобразити складну природу «нечітких» станів реальних метрик, недостатньо зупинитися на одномірному «пороговому» підході. Тому будується сукупність функцій належності, що задають лінгвістичні терми, такі як «Low», «Medium», «High» та «Very High», які утворюють гнучкий опис діапазонів зміни вхідних параметрів. Саме ця система функцій і дозволяє нечіткій логіці у подальшому коректно обробляти широкий спектр варіацій навантаження та формувати відповідні дії масштабування [110].

Для кожної з вхідних змінних (CPU та RAM) визначаються декілька нечітких лінгвістичних термів. Кількість рівнів може варіюватися залежно від вимог до точності й від того, наскільки великі перепади необхідно відображати. У системах із повільною зміною навантаження може бути достатньо трьох категорій, тоді як у динамічніших середовищах варто додати четверту, аби точніше зафіксувати критичний перехід за межу, де ризик деградації сервісу суттєво зростає.

Побудова функцій належності для згаданих термів ґрунтується на досвіді експертів з експлуатації систем чи даних, отриманих у результаті аналізу історичних трендів. Для побудови цих функцій доцільно використати реальні дані

споживання ресурсів, щоб відобразити типові режими навантаження. Для завантаженості CPU можна визначити кілька лінгвістичних термів: дуже низьке, низьке, середнє, високе та дуже високе використання CPU. Часто використовуються трикутні або трапецієвидні функції належності, рівномірно розподілені по діапазону можливих значень [111].

Нижче подано стандартну трапецієподібну (trapezoidal) функцію належності для нечіткої множини з параметрами a, b, c, d. Така функція зазвичай визначається як частково-лінійна та задає підйом від 0 до 1 між a і b, плато зі значенням 1 у відрізку $[b, c]$ та спад від 1 до 0 між c і d. Якщо вказані точки збігаються ($a = b$), то частина з підйомом вироджується, й можна отримати трикутну чи прямокутну форму. Для довільної змінної x у діапазоні [0,1] трапецієподібна функція $\mu_{trap}(x; a, b, c, d)$ описується такою функцією:

$$\mu_{trap}(x; a, b, c, d) = \begin{cases} 0, & x \le a, \\ \frac{x-a}{b-a}, & a < x < b, \\ 1, & b \le x \le c, \\ \frac{d-x}{d-c}, & c < x < d, \\ 0, & x \ge d. \end{cases} \quad (3.7)$$

Варто зауважити, що коли $a = b$, ліве плече стає вертикальним – не буде плавного підйому, а коли $c = d$, праве плече також вироджується у вертикальну лінію. Усі інші випадки легко отримуються з наведеної формули, коректно враховуючи граничні значення.

Функції належності вартості можна обрати трапецієвидними на кінцях діапазону, щоб відобразити, що дуже низькі витрати (близькі до нуля) і дуже високі (наближаються до ліміту) мають широкі області належності.

Вибір форм і кількості функцій належності впливає на ефективність системи та потребує налаштування. Немає єдиного стандарту побудови функцій належності – їх конфігурацію часто визначають експериментально або на основі експертних знань. На основі заданих нечітких правил система прийняття рішень визначає, коли

масштабувати (нарощувати або згортати ресурси) і наскільки змінювати кількість ресурсів. Нечіткий підхід дозволяє задавати гнучкі правила, що враховують одночасно декілька критеріїв [112].

Якщо у певній системі помічено, що якесь значення споживання трапляється дуже часто й при цьому не спостерігається зниження ефективності, функція належності «Low» може охоплювати більший діапазон значень і плавно переходити до функції «Medium». Коли ж споживання наближається до ліміту системи, AHPM прирівнює такий стан до «High» або навіть до «Very High», залежно від конкретних точок перетину й раніше зафіксованих трендів перевантаження.

Нижче наведені конкретні параметри (a, b, c, d) для чотирьох трапецієподібних функцій належності (усього діапазону від 0 до 1), які часто використовуються в задачах нечіткого оцінювання ступеня завантаження. Аргумент x тут можна розуміти як нормоване значення метрики RAM, тобто переведене в інтервал [0,1].

1. Low: $\mu_{Low}(x)$ із параметрами $a = 0, b = 0, c = 0.3, d = 0.35$.
2. Medium: $\mu_{Medium}(x)$ із параметрами $a = 0.3, b = 0.35, c = 0.6, d = 0.65$.
3. High: $\mu_{High}(x)$ із параметрами $a = 0.6, b = 0.65, c = 0.8, d = 0.85$.
4. Very High: $\mu_{VHigh}(x)$ із параметрами $a = 0.8, b = 0.85, c = 1.0, d = 1.0$.

У кожному випадку функція належності формується за описаною вище формулою. $\mu_{Low}(x)$ має спрощений вид через відсутність лівої бічної сторони:

$$\mu_{Low}(x) = \begin{cases} 1, & 0 < x \leq 0.3, \\ \frac{0.35 - x}{0.35 - 0.3}, & 0.3 < x < 0.35, \\ 0, & x \geq 0.35. \end{cases} \tag{3.8}$$

Аналогічно визначаються Medium, High і Very High з урахуванням своїх (a, b, c, d). Для кращої наочності можна побудувати чотири криві на одному графіку: кожна крива відповідає своїй функції належності (рис. 3.1).

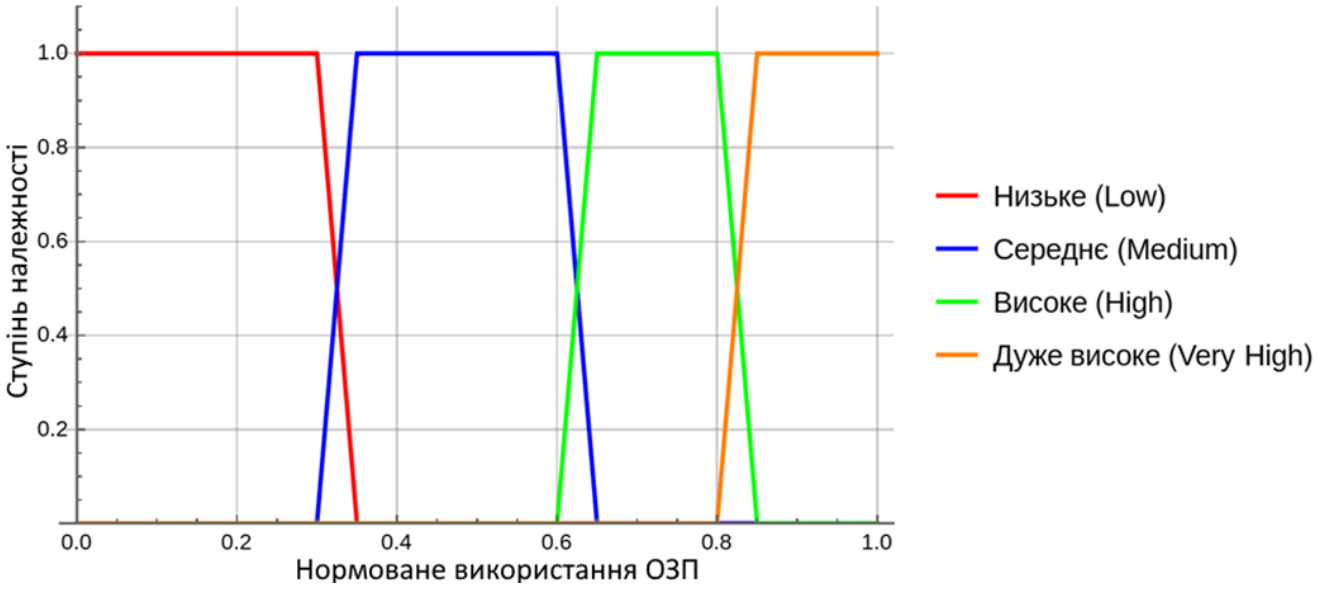

Рисунок 3.1. Функції належності використання оперативної пам’яті

За потреби можна увести ще один лінгвістичний терм, аби диференціювати високу й критично високу зони. Однак занадто велика кількість термів може ускладнити базу правил, тож слід прагнути до розумного компромісу між точністю й простотою. У разі виявлення в історії аномальних піків можна зсунути вправо параметри, що описують зону «High» і «Very High», аби надати більшої чутливості системі в цьому критичному діапазоні. Подібні коригування лежать в основі адаптивності: метод періодично аналізує статистику й зрушує положення функцій або змінює їх ширину, аби точніше інтерпретувати нові характеристики навантаження.

Якщо розглянути нормований варіант, тоді параметри функцій будуть такими:

1. Low CPU: $a = 0, b = 0, c = 0.19, d = 0.31$.

2. Medium CPU: $a = 0.19, b = 0.31, c = 0.56, d = 0.69$.

3. High CPU: $a = 0.56, b = 0.69, c = 0.8, d = 0.94$.

4. Very High CPU: $a = 0.81, b = 0.88, c = 1.0, d = 1.0$.

Ці функції зображені на рис. 3.2. Червоним кольором позначено функцію низького навантаження (Low), синім кольором – функцію середнього навантаження (Medium), зеленим – високого (High), а помаранчевим – функцію

дуже високого навантаження (Very High). Графік демонструє, що середнє значення для CPU ширше ніж для RAM. Це обумовлено тим, що для більшості систем завантаженість ЦП на 30-50% є нормою і масштабування не потрібне.

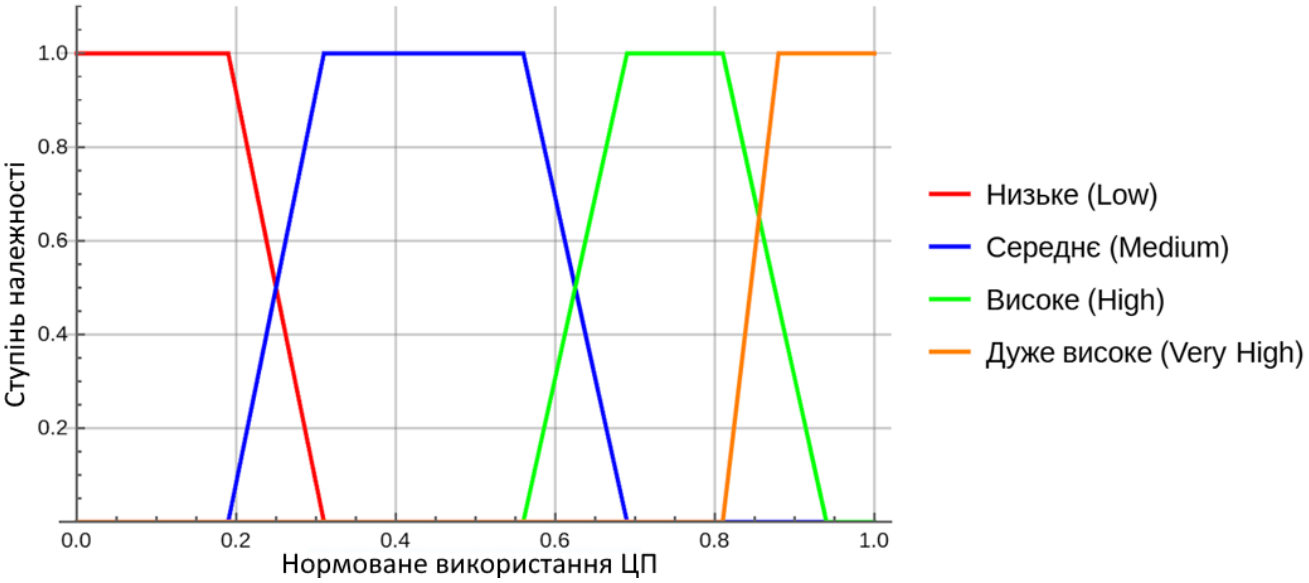

Рисунок 3.2. Функції належності завантаженості процесора

Таким чином, λ фазифікується у набір пар значень виду $\{(\tilde{L}, \mu(\tilde{L}))\}$. Для побудови адаптивної нечіткої моделі динамічного масштабування ресурсів було визначено універсум можливих значень для основних метрик системи без жорсткої прив'язки до конкретних апаратних лімітів. Це забезпечує гнучкість відповідного методу та дозволяє масштабувати систему незалежно від змін у конфігурації апаратних ресурсів. Вибір такого універсуму дозволяє методу адаптуватися до зміни доступних ресурсів (збільшення кількості процесорних ядер або розширення оперативної пам'яті), оскільки нечіткі правила залишаються актуальними, а функції належності автоматично масштабуються. Для кожної метрики було сформовано чотири нечіткі множини, що дозволяють коректно враховувати перехідні стани та забезпечувати гнучкість у масштабуванні. Використано трапецієподібні функції належності, які дозволяють нечітко визначати межі між рівнями завантаження.

3.2.4 Побудова таблиці коефіцієнтів

Отримані нечіткі значення після етапу фазифікації підлягають подальшій обробці за допомогою спеціально побудованої таблиці коефіцієнтів. Побудова такої таблиці є одним із центральних етапів розробки адаптивної нечіткої регресивної моделі динамічного масштабування ресурсів. На відміну від класичних продукційних нечітких систем, де висновок здійснюється через використання чітко сформульованих правил, запропонований підхід передбачає створення таблиці коефіцієнтів, які відображають ступінь впливу відповідних комбінацій нечітких термів вхідних параметрів на вихідні керуючі значення.

Необхідність побудови таблиці коефіцієнтів зумовлена складністю і різноманітністю реальних сценаріїв навантаження на ресурси (табл. 3.4).

Таблиця 3.4

Матриця ваг нечітких термів завантаженості CPU

Міри впливу	Low RAM	Medium RAM	High RAM	VeryHigh RAM
Low CPU	0.125	0.125	0.1875	0.1875
Medium CPU	0.25	0.3125	0.3125	0.375
High CPU	0.5	0.5625	0.625	0.6875
VeryHigh CPU	0.9	0.95	0.95	1

В умовах динамічних систем навантаження на центральний процесор і пам’ять не завжди змінюється незалежно, часто спостерігається певна кореляція [113].

Водночас ступінь цього взаємного впливу може бути різним, тому класичне формулювання правил у вигляді жорстких продукцій може призводити до надмірної спрощеності або, навпаки, до значного ускладнення системи.

Використання таблиці коефіцієнтів, які визначають вплив кожної комбінації нечітких термів, дозволяє більш гнучко врахувати не лише основний вплив кожного ресурсу окремо, а й слабкі взаємні впливи між ними (табл. 3.5).

Таблиця 3.5

Матриця ваг нечітких термів завантаженості RAM

Міри впливу	Low CPU	Medium CPU	High CPU	VeryHigh CPU
Low RAM	0.125	0.1375	0.15	0.1625
Medium RAM	0.25	0.275	0.3	0.325
High RAM	0.5	0.5375	0.575	0.6125
VeryHigh RAM	0.8	0.9	0.9	1

Процес побудови таблиці коефіцієнтів розпочинається з визначення вхідних змінних, які описуються стандартними нечіткими термами. Далі, на основі експертного знання та аналізу поведінки реальних систем формуються початкові коефіцієнти для кожної комбінації термів. Ці коефіцієнти відображають рекомендовані значення нормованого керуючого параметра масштабування ресурсів, враховуючи основний вплив одного ресурсу та слабкий, але важливий вплив іншого.

Однією з ключових особливостей таблиці коефіцієнтів є те, що вона дозволяє уникнути використання великої кількості окремих параметрів і ваг, які потрібно було б налаштовувати окремо. Натомість експертні знання одразу закладаються в єдину таблицю, що істотно спрощує процедуру налаштування методу та робить його більш зрозумілим та інтерпретованим. При цьому коефіцієнти таблиці обираються так, що вони враховують не тільки базові оцінки навантаження окремих ресурсів, а й ті особливі ситуації, що можуть виникати в системі.

Коли навантаження CPU є високим, а RAM – низьким, це може свідчити про короткочасний сплеск обчислювальної активності, який з високою ймовірністю не потребує масштабування CPU до дуже високих значень, а скоріше сигналізує про тимчасовий характер такого навантаження.

Запропонована таблиця коефіцієнтів замінює класичну базу правил, оскільки кожен коефіцієнт відповідає певному правилу з нечіткою передумовою та чітким числовим наслідком. Проте, на відміну від класичних продукційних моделей, у запропонованій моделі немає явного формулювання правил типу «якщо–то». Натомість нечіткі передумови для кожної комбінації термів, отримані на етапі фазифікації, агрегуються через ступінь належності, а таблиця коефіцієнтів визначає значення, яке відповідає кожній комбінації.

Модель зберігає свою регресивну природу, адже числовий вихід формується не через просту дискретну логіку, а через неперервне зважене агрегування нечітких оцінок [114]. Використання таблиці коефіцієнтів в адаптивній динамічній нечіткій регресивній моделі масштабування ресурсів є обґрунтованим та ефективним рішенням, що дозволяє поєднати переваги нечіткої логіки та регресивного аналізу.

На рис 3.3 показано візуалізацію матриць ваг нечітких термів у вигляді контурних діаграм.

Контурні діаграми наочно демонструють залежність рекомендацій щодо масштабування ресурсів від двох вхідних параметрів: ступеня завантаження процесора (CPU) та оперативної пам'яті (RAM), які визначені нечіткими термінами (низьке, середнє, високе, дуже високе).

Осі X та Y відображають відповідно ступінь завантаженості процесора та пам'яті. Контури поверхні показують плавні переходи між різними станами навантаження, ілюструючи, як нечітка логіка дозволяє уникнути різких змін у масштабуванні ресурсів, забезпечуючи тим самим плавні та прогнозовані переходи між станами завантаження, що допомагає чітко бачити залежність прийняття рішень від одночасного впливу CPU та RAM, підкреслюючи гнучкість та адаптивність AHPM.

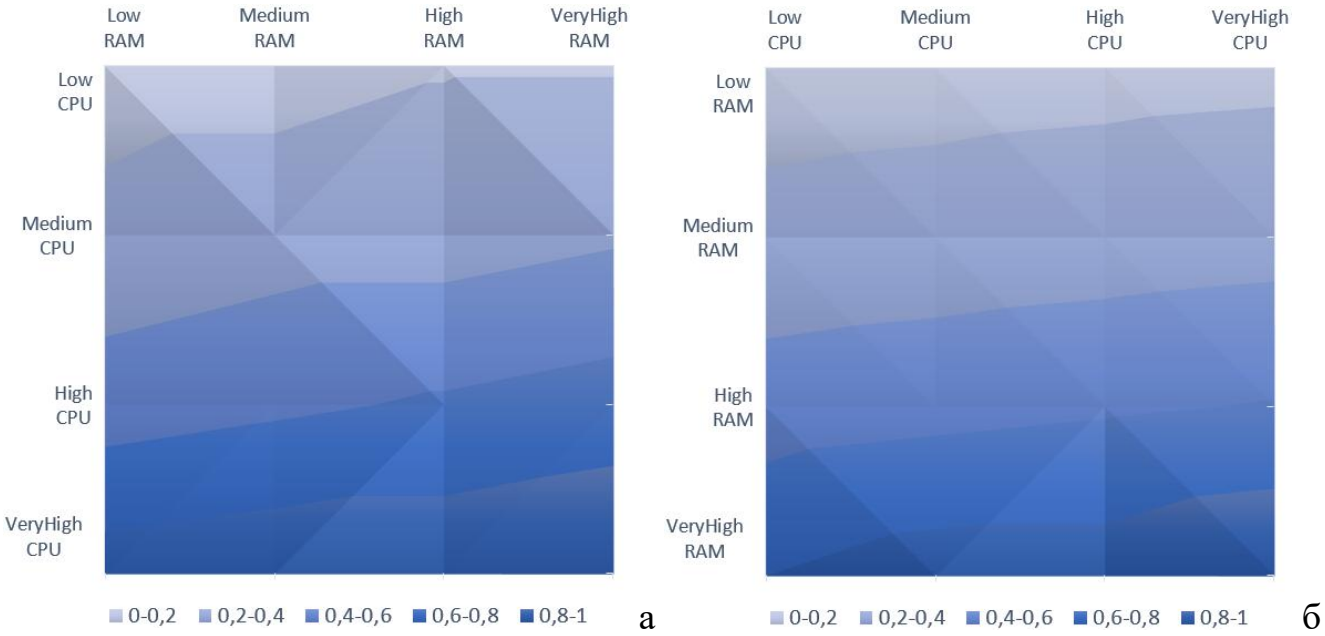

Рисунок 3.3. Коефіцієнти для регресивного оцінювання CPU (а), RAM (б)

Такий підхід зберігає регресивну природу AHPM, забезпечує високу інтерпретованість, гнучкість, простоту налаштування і водночас дає змогу враховувати тонкі особливості взаємодії ресурсів у динамічних системах із високим рівнем невизначеності.

3.2.5 Агрегація правил та дефазифікація

Етап агрегації правил є важливою складовою процесу нечіткого висновку в адаптивній нечіткій регресивній моделі динамічного масштабування ресурсів. Цей етап забезпечує перехід від набору часткових нечітких рекомендацій, що відповідають різним комбінаціям термів завантаженості ресурсів, до єдиного числового значення, придатного для подальшого використання у прийнятті управлінських рішень щодо масштабування.

Оскільки в запропонованій моделі явно не формулюються класичні правила типу «якщо–то», агрегація здійснюється шляхом зважування коефіцієнтів, визначених на попередньому етапі в таблиці коефіцієнтів, на відповідні ступені належності вхідних нечітких значень до лінгвістичних термів. Використання

множення ступенів належності забезпечує плавний перехід між різними сценаріями навантаження, що відображає реальну динамічну природу поведінки високонавантажених систем.

Разом з тим, хоч формально етап дефазифікації відсутній, з практичної точки зору запропонований підхід можна інтерпретувати як інтегровану дефазифікацію. Завдяки використанню числових коефіцієнтів у таблиці коефіцієнтів, кожна нечітка оцінка одразу проектується на конкретне числове значення, а зважена сума цих значень дає неперервний числовий результат, готовий до подальшого використання. При зміні умов експлуатації системи або появі нових сценаріїв навантаження адаптація АНРМ здійснюється просто через коригування коефіцієнтів у таблиці. Це робить запропонований метод перспективним для практичного застосування в автоматизованих системах керування розподіленими обчислювальними ресурсами.

Остаточні керуючі показники Y_{cpu} та Y_{ram} формуються як зважена сума ступенів належності нечітких термів відповідного ресурсу, де вплив іншого ресурсу враховується через домінуючий терм. Використання домінуючого нечіткого терма дає змогу виділити основну характеристику стану ресурсу.

Якщо при фазифікації показника навантаження RAM кілька нечітких термів (Low, Medium, High, VeryHigh) мають певні ступені належності, то домінуючий терм – той, для якого ступінь належності максимальний – найбільш точно відображає поточний стан ресурсу. На відміну від класичних нечітких систем, де враховується повна агрегація всіх нечітких входів, використання домінуючого терма зменшує обчислювальну складність та підвищує стабільність АНРМ, що є вагомим внеском у розвиток методів адаптивного масштабування ресурсів.

Нехай для ресурсу ступені належності до нечітких термів визначаються як $\mu(\tilde{L}_1),\ \mu(\tilde{L}_2),\ \mu(\tilde{L}_3),\ \mu(\tilde{L}_4)$. Позначимо через i^* та j^* індекси, для яких ступенів належності до ресурсу максимальний, при цьому, якщо максимум досягається для декількох індексів, застосовуємо правило розв’язання рівності і вибираємо найменший індекс:

$$i^* = \min\left\{i \in \{1,2,3,4\} \mid \mu_{cpu}\left(\tilde{L}_i^{cpu}\right) = \max_{r\in\{1,2,3,4\}} \mu_{cpu}\left(\tilde{L}_r^{cpu}\right)\right\} \quad (3.9)$$

$$j^* = \min\left\{j \in \{1,2,3,4\} \mid \mu_{ram}\left(\tilde{L}_j^{ram}\right) = \max_{r\in\{1,2,3,4\}} \mu_{ram}\left(\tilde{L}_r^{ram}\right)\right\} \quad (3.10)$$

Формули керуючих показників дозволяють отримати неперервні керуючі показники на основі нечіткої регресії.

$$Y_{cpu} = \sum_{i=1}^{4} \mu_{cpu}\left(\tilde{L}_{ij}^{cpu}\right) \cdot k_{cpu}\left(\tilde{L}_i^{cpu}, \tilde{L}_{j^*}^{ram}\right) \quad (3.11)$$

$$Y_{ram} = \sum_{j=1}^{4} \mu_{ram}\left(\tilde{L}_j^{ram}\right) \cdot k_{ram}\left(\tilde{L}_j^{ram}, \tilde{L}_{i^*}^{cpu}\right), \quad (3.12)$$

де:

Y_{cpu}, Y_{ram} – нормовані керуючі показники для масштабування;

μ – функція визначення належності поточного навантаження;

k – функція визначення коефіцієнту, що вибирається з матриці ваг нечітких термів.

Для остаточного прийняття рішення щодо виділення ресурсів ці показники мають бути денормалізовані з урахуванням системних лімітів.

Для оперативної пам’яті (RAM) денормалізація здійснюється прямо: Y_{ram} множиться на ліміт системи R, тобто:

$$N_{ram} = Y_{ram} \cdot R, \quad (3.13)$$

де N_{ram} – остаточна кількість RAM, яка має бути виділена.

Процесорні ресурси вимагають більш тонкого підходу, оскільки у системах з високою продуктивністю часто важливо не перевантажувати CPU, щоб уникнути затримок і забезпечити високий рівень продуктивності, а з іншого боку — враховувати фінансові обмеження. Для цього введено коефіцієнт оптимальності, який регулюється залежно від бажаного рівня завантаженості процесора.

Якщо оптимальна завантаженість визначається як 80 % коефіцієнт дорівнюватиме $\frac{100}{80}$, а якщо уникнення затримок і підвищення продуктивності важливіше ніж фінанси і зростання вартості, то оптимальною завантаженістю процесора може бути 20%, тоді коефіцієнт буде дорівнювати $\frac{100}{20}$.

Цей коефіцієнт коригує вихідну рекомендацію, адже при нижчій оптимальній завантаженості система повинна виділяти більше процесорних ресурсів для забезпечення високої продуктивності, а при вищій завантаженості – навпаки, щоб не перевантажувати систему. Таким чином, остаточну кількість CPU розраховують за формулою

$$N_{cpu} = f_{opt} \cdot Y_{cpu} \cdot C \quad \text{де} \quad f_{opt} = \frac{100}{\text{оптимальний відсоток завантаженості } CPU} \quad (3.14)$$

Використання коефіцієнта f_{opt} є необхідним, оскільки системи з різними обмеженнями та експлуатаційними умовами можуть мати різні оптимальні рівні завантаженості процесора. Також це дозволяє методу адаптуватися до конкретних вимог та умов експлуатації, що є суттєвою науковою новизною, оскільки він інтегрує оптимізаційний аспект у процес денормалізації керуючих показників.

На рис. 3.4 зображено блок-схему усіх етапів роботи методу.

Експертно визначено, що оптимальний рівень навантаження CPU становить 33 %, тому коефіцієнт f_opt=3. АНРМ ефективно апроксимує кількісну залежність між вхідними показниками навантаження (представленими нечіткими термами) та вихідними керуючими величинами.

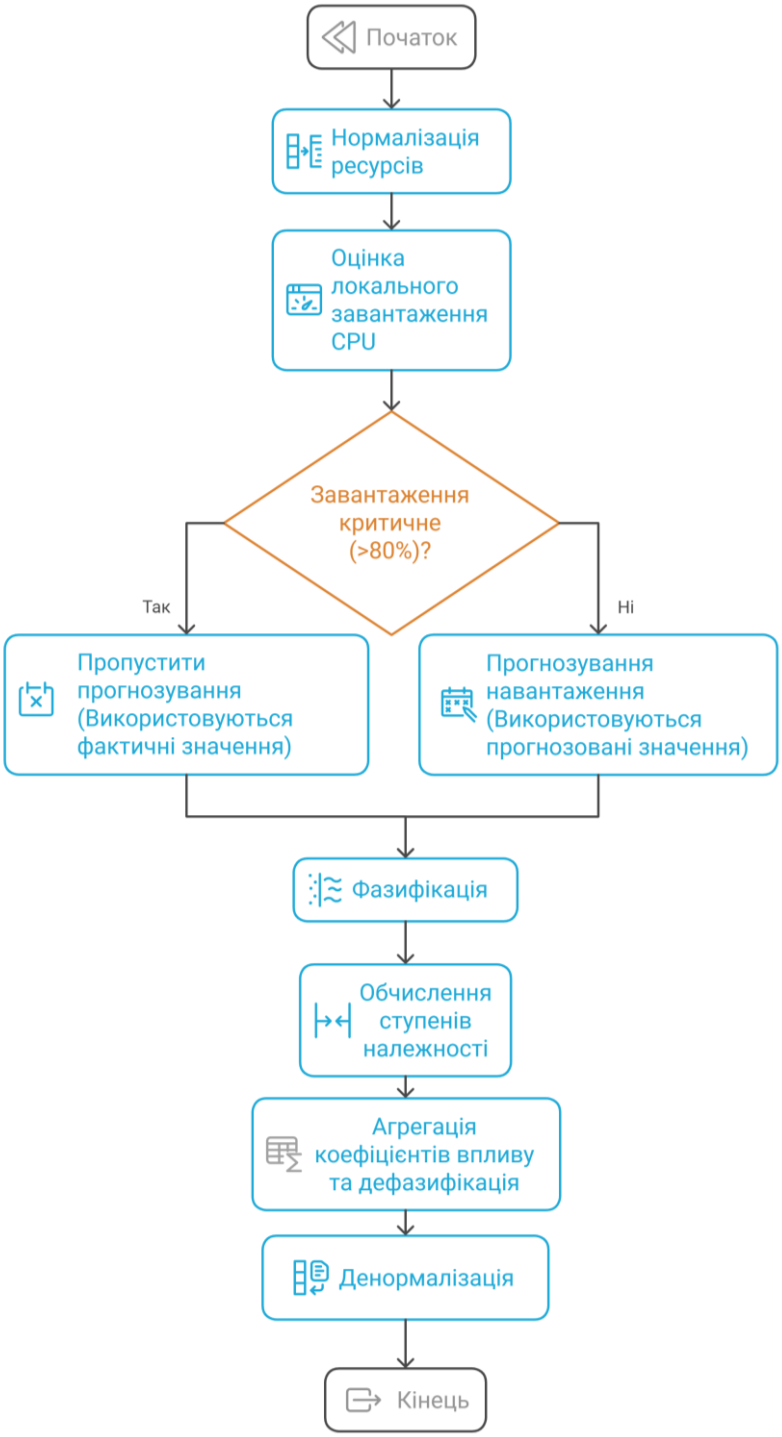

Рисунок 3.4. Блок-схема послідовності етапів рахування AHPM

3.3. Формування методу коригування моменту масштабування

AHPM прогнозує сам пік навантаження, визначаючи його час початку, тривалість та амплітуду. Однак одного лише прогнозу піку недостатньо для ефективного використання ресурсів.

Додатковий метод коригування моменту масштабування аналізує:

1. Чи завчасно було запущено масштабування (перевірка на простоювання ресурсів).

2. Чи запізно було виконано зменшення ресурсів (чи була система завантажена довше, ніж потрібно).

3. Оптимізує моменти старту та згортання масштабування на основі історичних даних.

Нижче наведено опис роботи методу коригування, який ретроспективно аналізує дані прогнозування навантаження та, на основі отриманих похибок, коригує коефіцієнти AHPM. Система, яка керує ресурсами використовуючи AHPM, не має чітких дискретних етапів запуску чи завершення масштабування; натомість вона постійно регулює рівень ресурсів.

Вхідними даними для методу є історичні часові інтервали (початок і кінець періоду), коли система знаходилась у режимах піку, коли нормоване завантаження CPU, λ_{cpu}, перевищувало 0.8 та коли система була практично неактивною – $\lambda_{cpu} < 0.2$. Крім того, зберігаються часові значення, коли система здійснювала масштабування (t_{scale}). Метод аналізує, як співвідносяться ці моменти з прогнозованими значеннями.

Нехай для кожного інциденту s із історичних даних визначаються наступні часові моменти: $t_{\text{crit}}^{\uparrow,(\text{s})}$ – момент, коли система перейшла у критичний режим (коли $\lambda_{cpu} > 0.8$), $t_{\text{crit}}^{\downarrow,(\text{s})}$ – момент, коли система перейшла у режим профіциту (коли $\lambda_{cpu} < 0.2$), t_{scale} – момент, коли було здійснено зміну рівня ресурсів (масштабування).

Для кожного такого інциденту можна визначити похибку прийняття управлінського рішення, як різницю між t_{scale} і $t_{\text{crit}}^{\uparrow,(\text{s})}$ для масштабування вгору або між t_{scale} та $t_{\text{crit}}^{\downarrow,(\text{s})}$ для масштабування вниз. Метод спочатку агрегує ці дані за всіма N інцидентами, розраховуючи середнє відхилення для фаз збільшення та зниження ресурсів. Нехай $T_{scale\_lead} = \frac{1}{N}\sum_{s=1}^{N}(t_{\text{crit}}^{\uparrow,(\text{s})} - t_{scale}^{(s)})$ відображає середню величину

випередження, а $T_{scale\_lag} = \frac{1}{N}\sum_{s=1}^{N}(t_{scale}^{(s)} - t_{\text{crit}}^{\downarrow,(s)})$ характеризує середню затримку завершення масштабування. Ці середні величини показують середню величину випередження та середню затримку масштабування.

Середнє абсолютне відхилення часу масштабування визначається за (3.15):

$$C = \frac{|T_{scale\_lead}|+|T_{scale\_lag}|}{2} \tag{3.15}$$

Далі обчислюється корекційний крок як:

$$\Delta = \min\{k_c \cdot \mathrm{C}, \Delta_{max}\}, \tag{3.16}$$

де k_c – пропорційна константа (0.01), а Δ_{max} – максимальне допустиме змінення (0.1).

Оновлення коефіцієнтів проводиться за умовами: якщо $T_{scale\_lead} < 0$ та $T_{scale\_lag} > 0$, то це означає, що масштабування відбувається із запізненням (ресурси активувалися пізніше оптимального моменту або зниження ресурсів відбувається із затримкою).

Тоді для швидшої реакції збільшується α, а β та γ зменшуються:

$$\begin{aligned} \alpha_{new} &= \min\{1, \max\{0, \alpha_{old} + \Delta\}\}; \\ \beta_{new} &= \min\{1, \max\{0, \beta_{old} - \Delta\}\}; \\ \gamma_{new} &= \min\{1, \max\{0, \gamma_{old} - \Delta\}\}. \end{aligned} \tag{3.17}$$

Якщо $T_{scale\_lead} > 0$ та $T_{scale\_lag} < 0$, то це свідчить, що масштабування відбувається занадто рано (ресурси активуються раніше оптимального моменту або деактивація ресурсів відбувається передчасно).

У такому випадку α зменшується, а β та γ збільшуються:

$$\begin{aligned} \alpha_{new} &= \min\{1, \max\{0, \alpha_{old} - \Delta\}\}; \\ \beta_{new} &= \min\{1, \max\{0, \beta_{old} + \Delta\}\}; \\ \gamma_{new} &= \min\{1, \max\{0, \gamma_{old} + \Delta\}\} \end{aligned} \tag{3.18}$$

Якщо умови для корекції не виконуються (тобто, якщо середні відхилення не свідчать про систематичну помилку), коефіцієнти залишаються незмінними. Таким чином, метод коригування використовує історичні значення T_{scale_lead} та T_{scale_lag} для визначення середньої тривалості відхилень, обчислює корекційний крок Δ пропорційно до цих відхилень (з обмеженням максимальної величини), а потім оновлює коефіцієнти α, β та γ за наведеними правилами.

На рис. 3.5 зображено взаємодію методів прогнозування та коригування.

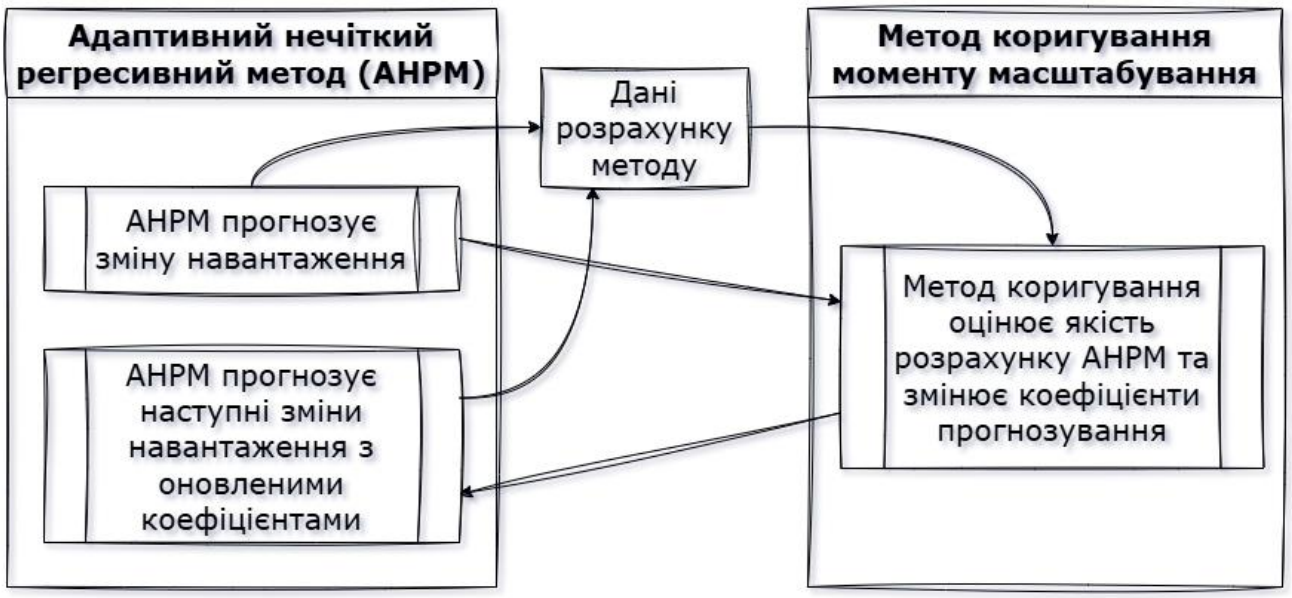

Рисунок 3.5. Схема взаємодії АНРМ та методу коригування моменту масштабування

АНРМ дозволяє ефективно коригувати параметри прогнозування без використання складних алгоритів і методів, забезпечуючи більш точну адаптацію системи масштабування ресурсів до реальних умов роботи. Також метод є адаптивним і дозволяє АНРМ самостійно налаштовуватись, що підвищує

ефективність управління ресурсами без залучення складних моделей машинного навчання.

3.4. Багатокритеріальна оптимізація балансу вартості й продуктивності

У задачі автоматичного масштабування ресурсів для високонавантажених систем важливо не лише забезпечити необхідний рівень продуктивності, а й мінімізувати витрати на інфраструктуру. Оптимальне рішення повинно враховувати компроміс між якістю управління ресурсами та економічною ефективністю. Google Autoscaler є комерційним рішенням з передовими методами машинного навчання, а AHP використовує експоненційне згладжування та нечітку логіку для балансування ресурсів.

Для кількісного порівняння ефективності масштабування використовуємо метрику, яка визначає скільки часу був дефіцит ресурсів при піковому навантаженні: $Q_{deficit} = \frac{|T_{scale\_lag}|}{\max\{t_{crit}\}}$.

Також метрику, яка визначає скільки часу було виділено більше ресурсів, ніж потрібно при піковому навантаженні: $Q_{waste} = \frac{|T_{scale\_lead}|}{\max\{t_{crit}\}}$.

Врешті, об’єднуючи обидві метрики, отримуємо загальну ефективність масштабування: $Q_{efficiency} = 1 - (Q_{deficit} + Q_{waste})$. Дана метрика ефективності відображає баланс між дефіцитом і перевитратою ресурсів.

Отже формула, що визначає умови вигідності AHPM у порівнянні з Google Autoscaler, має вигляд:

$$Q_{\text{AHPM},avg} \geq Q_{Google,avg} \times \frac{C_{\text{AHPM}}}{C_{Google}}, \tag{3.19}$$

де:

- $Q_{\text{AHPM},avg}$ – середня ефективність AHPM,
- $Q_{Google,avg}$ – середня ефективність Google Autoscaler,
- C_{AHPM} – вартість AHPM на годину,

– C_{Google} – вартість Google Autoscaler на годину.

Якщо $C_{\text{AHPM}} = 0$, то AHPM є вигіднішою, якщо її ефективність хоча б трохи більше за 0%. Якщо $C_{\text{AHPM}} = C_{Google}$, то AHPM повинна мати таку ж ефективність або вищу, інакше вона програє за всіма параметрами. Але якщо $C_{\text{AHPM}} < C_{Google}$, то AHPM може бути менш ефективною, але все ще вигіднішою завдяки нижчій ціні. Ця формула дозволяє встановити максимально допустиме значення вартості AHPM залежно від її ефективності.

Якщо за деякий період роботи Google Autoscaler помиляється і допускає випадки, коли масштабування відбувалося занадто рано, що призводило до нераціонального використання ресурсів і збільшення витрат або випадки, коли масштабування відбувалося занадто пізно, що призводило до зайвих витрат на невикористані ресурси, то його показник ефективності ($Q_{Google,avg}$) падає.

Вартість 1 GKE Autopilot vCPU становить $0.0685/год або $50/місяць для General-purpose (default) Pods в регіоні найближчому до України – Berlin (europe-west10) [115].

На рис. 3.6 зображено 4 функції:

1. Вартість без масштабування – збільшується щогодини на вартість 8CPU/16RAM. Це описує систему в якій не встановлено автоматичний масштабувальник і вона працює завжди на максимумі доступних ресурсів.

2. Вартість Google Autoscaler – також збільшується щогодини, але коли якість роботи масштабувальника знижується, вартість додатково росте, адже були простої ресурсів і вартість утримання системи виросла.

3. Якість роботи Google Autoscaler – бере початок зі 100% і має тенденцію до зниження до ~80%.

4. Якість роботи AHPM – також бере початок зі 100% і має тенденцію до зниження до ~50%.

Google Autoscaler демонструє високу якість масштабування, що стартує з 100% і поступово знижується до ≈80% через затримки в реакції на динамічні зміни

навантаження. AHPM також починає з 100% якості, але через спрощені підходи поступово знижується до 50%.

Обчислимо середнє значення ефективності за 24 години роботи:

- Google Autoscaler: середня ефективність $Q_{Google,avg} = 0.9$
- AHPM: середня ефективність $Q_{\text{AHPM},avg} = 0.75$

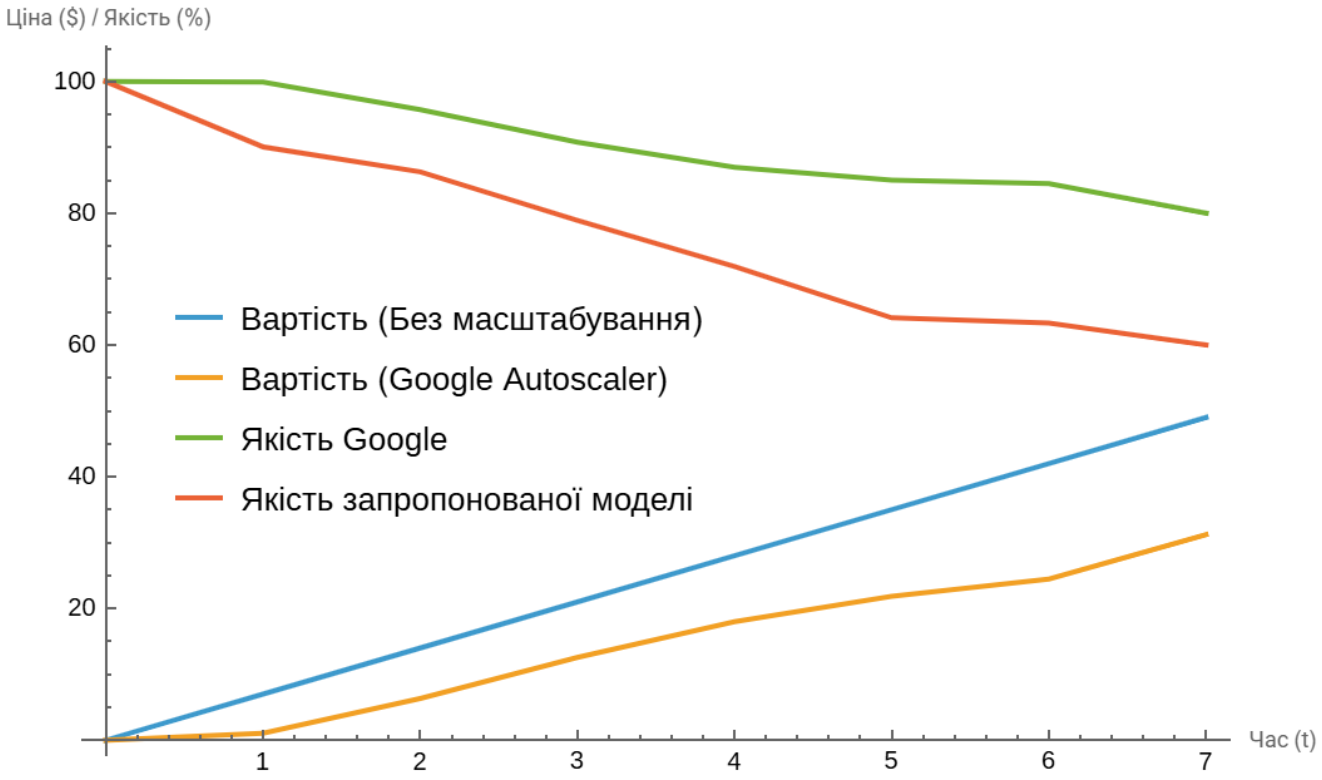

Рисунок 3.6. Оптимізація співвідношення ціни та ефективності масштабування

Вартість Google Autoscaler на годину складає \$0.0685. Тому його індекс ефективності: $E_{Google} = \frac{0.9}{0.0685} = 13.14$, тому щоб AHPM залишалася ефективнішою за Google Autoscaler, її максимальна допустима вартість на годину вираховується як: $C_{\text{AHPM},max} = \frac{Q_{\text{AHPM},avg}}{E_{Google}} = \frac{0.75}{13.14} = 0.057$.

Це означає, що AHPM залишається вигіднішою, якщо її вартість не перевищує \$0.057 на годину, що всього менш ніж на 16% нижче ціни Google. Тож якщо користувач готовий трохи поступитися якістю масштабування заради

суттєвого зменшення витрат, то використання АНРМ є економічно вигіднішим вибором. За рахунок гнучкого налаштування вартості використання, АНРМ може бути впроваджено як бюджетну альтернативу комерційним рішенням, забезпечуючи оптимальний баланс між ефективністю та вартістю.

Ще однією вагомою перевагою АНРМ є її платформна незалежність. Використовуючи комерційне рішення, розробники змушені приймати правила ціноутворення та архітектурні обмеження певного хмарного провайдера, що може суттєво впливати на вартість експлуатації [116-119]. АНРМ може бути розгорнуто в будь-якому середовищі, незалежно від хмарного провайдера чи інфраструктури.

Завдяки цьому АНРМ не тільки дозволяє оптимізувати баланс між вартістю та продуктивністю, але й дає користувачам повний контроль над вибором інфраструктури, що робить її більш адаптивною та економічно вигідною порівняно з прив’язаними до конкретних платформ комерційними рішеннями.

Висновки до розділу 3

1. Запропоновано адаптивну нечітку регресивну модель динамічного масштабування ресурсів високонавантажених розподілених систем. На основі цієї моделі розроблено адаптивний нечіткий регресивний метод, що забезпечує гнучке керування ресурсами без жорстких порогових значень. У межах методу побудовано систему нечіткого виведення та сформовано матрицю ваг термів; доведено, що використання домінуючого терма знижує обчислювальну складність і підвищує стабільність оцінювання.

2. Розроблено метод коригування моменту масштабування, який аналізує вихідні величини нечітко-регресивного методу й оцінює наслідки запізнення або передчасної реакції. Запропонований метод коригує параметри прогнозування, що підвищує точність передбачення зміни навантаження та забезпечує раціональний час оновлення конфігурації ресурсів.

3. Інтегровано експоненційне згладжування з подвійною корекцією (за похибкою прогнозу та трендом) у нечітку регресивну модель; формалізовано взаємодію між прогнозною складовою та нечіткою логікою. Показано, що така інтеграція зменшує затримку ухвалення рішень і підвищує точність прогнозу моментів пікових навантажень і необхідного рівня ресурсів.

РОЗДІЛ 4. ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ АДАПТИВНОГО НЕЧІТКОГО РЕГРЕСИВНОГО МЕТОДУ МАСШТАБУВАННЯ РЕСУРСІВ І МЕТОДУ КОРИГУВАННЯ МОМЕНТУ МАСШТАБУВАННЯ

У розділі проаналізовано результати експериментальних досліджень, які підтвердили ефективність запропонованого адаптивного нечіткого регресивного методу динамічного масштабування ресурсів, що поєднує експоненційне згладжування для прогнозу навантаження з нечітким агрегуванням лінгвістичних оцінок; метод дає змогу своєчасно виділяти ресурси, уникати фіксованих порогів і забезпечувати стабільну роботу високонавантажених розподілених систем за оптимального використання обчислювальних потужностей; на прикладі геоінформаційного модуля системи відновлення пошкоджених будівель продемонстровано переваги проактивного масштабування, завдяки якому короткочасні пікові періоди нівелювалися завчасним збільшенням ресурсів і подальшим поверненням до економнішого режиму; для інтелектуальної системи керування транспортним трафіком великого міста показано, що повторюваність годин «пік» полегшує прогнозування, а метод адаптивно коригує обсяг ресурсів під поточний рівень навантаження; на високонавантаженій платформі аналізу даних із соціальних мереж підтверджено здатність методу плавно масштабувати ресурси при поступовому довготривалому зростанні користувацької активності; запропонований метод може бути інтегрований у практичні системи з високою варіативністю навантаження, забезпечуючи гнучкий і своєчасний розподіл обчислювальних потужностей.

4.1. Вдосконалення геоінформаційного модуля системи підтримки процесу відновлення будівель, споруд і об’єктів інфраструктури

На рис. 4.1 показано архітектуру інформаційно-комунікаційної системи підтримки процесу відновлення будівель, споруд і об’єктів інфраструктури (СППВОН) [120-122].

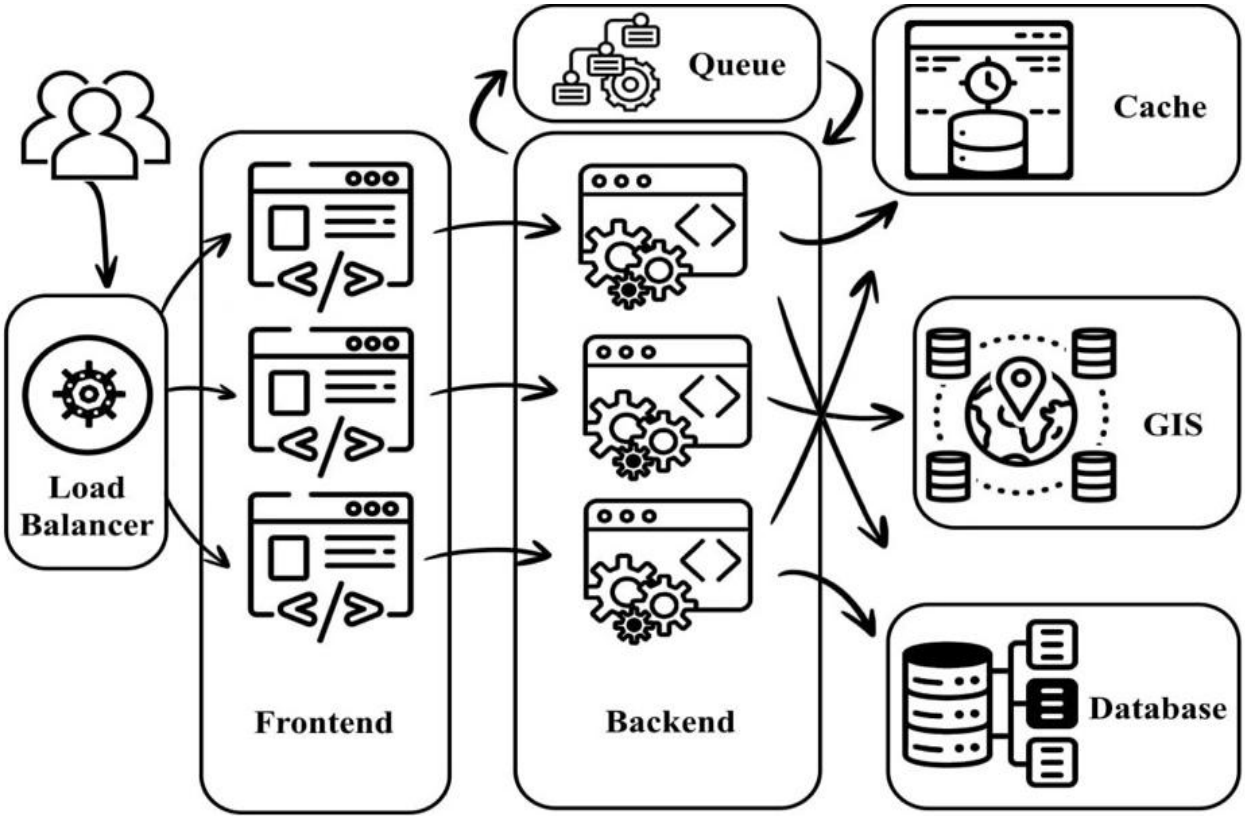

Рисунок 4.1. Архітектура СППВОН [121]

СППВОН призначена для вирішення багатьох задач високонавантаженою розподіленою системою, користувачами є громадяни, громади, фахівці галузі, експерти, інвестори, центри надання адміністративних послуг, будівельні фірми, органи державної влади, органи соціального захисту населення, органи досудового розслідування і спеціалізовані фонди відновлення України [120].

В роботі [122] запропоновано і детально описано модуль обробки і збереження даних СППВОН, що призначений забезпечити високу продуктивність, масштабованість, надійність і безпеку розподіленій системі підтримки процесу відновлення об’єктів нерухомості в умовах високих навантажень.

Діаграма послідовності зображена на рис. 4.2 демонструє взаємодію між сервісами та компонентами інфраструктури. Кожен запит від користувача проходить через сервіс маршрутизації і запускає відповідні дії в сервісах автентифікації, даних і геоінформації, а логування відбувається через брокер повідомлень і відповідний сервіс.

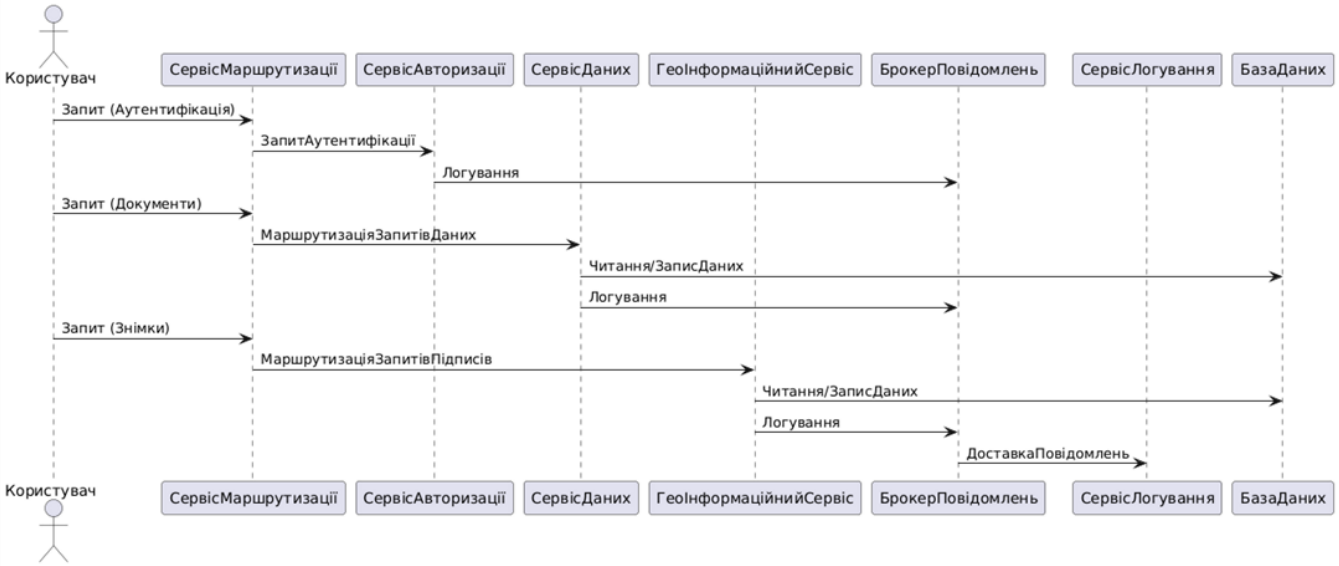

Рисунок 4.2. Діаграма послідовності взаємодії мікросервісів [120]

Універсальну схему мікросервісної архітектури модуля обробки і збереження даних, що уніфікує підхід до маніпуляції даними, забезпечуючи ефективну інтеграцію різних даних із різних джерел інформації показано на рис. 4.3.

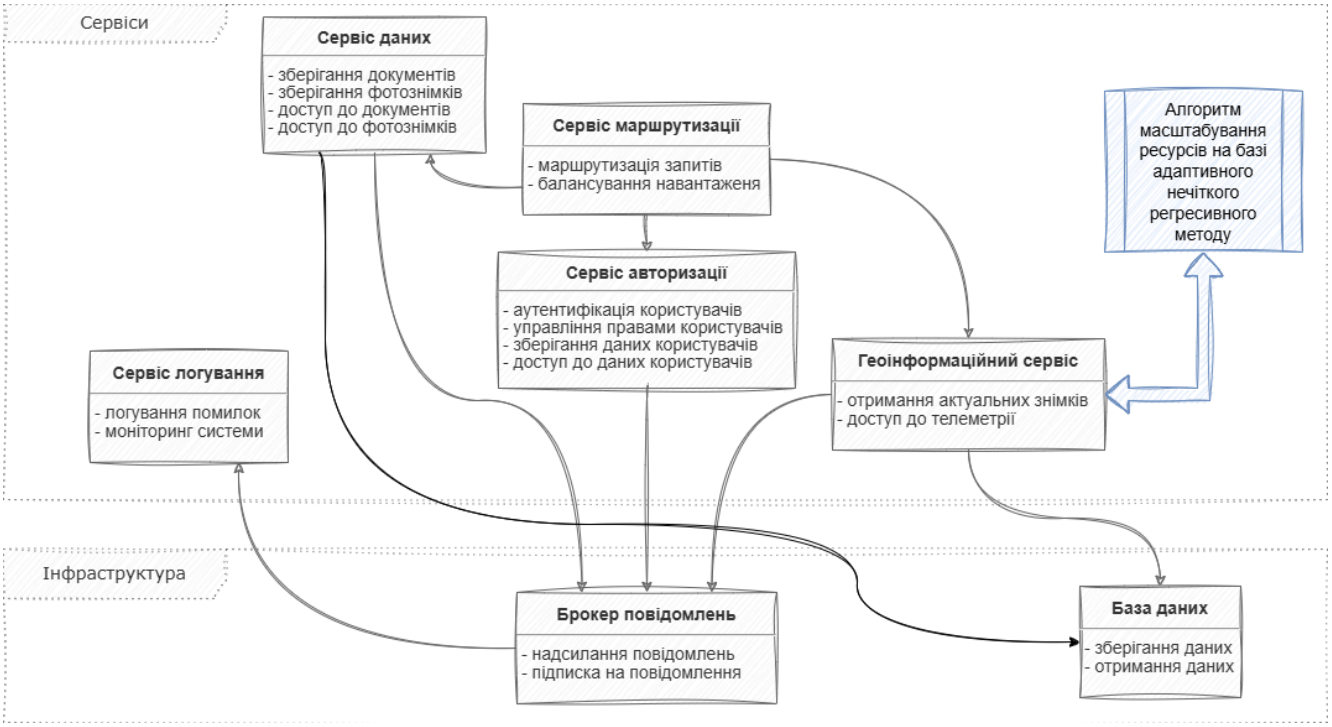

Рисунок 4.3. Схема мікросервісної архітектури модуля обробки і збереження даних СППВОН [121]

Взаємодія між цими компонентами забезпечує комплексний підхід до управління даними, безпеки та продуктивності системи. Така архітектура дозволяє легко впроваджувати нові функціональні можливості, тестувати покращення та забезпечувати високу продуктивність у різних сценаріях використанняказано схему комунікації СППВОН з зовнішнім середовищем.

Взаємодія між компонентами модуля забезпечує комплексний підхід до управління даними, безпеки і продуктивності СППВОН.

Кожен компонент модуля, взаємодіючі з іншими, виконує свою специфічну роль для забезпечення безперебійної роботи і високої ефективності системи в цілому, а саме:

- сервіс авторизації, що відповідає за автентифікацію користувачів і управління їх правами доступу, забезпечує безпеку системи, запобігаючи несанкціонованому доступу до ресурсів і даних;

- сервіс маршрутизації, що діє як єдина точка входу для всіх клієнтських запитів, забезпечує маршрутизацію запитів; при цьому балансування навантаження підвищує загальну продуктивність системи і запобігає перевантаженню окремих компонентів, забезпечуючи рівномірний розподіл запитів між сервісами;

- сервіс даних зберігає інформацію про користувачів і документи, а також забезпечує доступ до цих даних; цей сервіс є критично важливим для системи, оскільки об'єднує всю основну бізнес-логіку і дані, що зберігаються в системі;

- сервіс логування збирає і зберігає інформацію про всі помилки і збої, які виникають в системі; це важливо для моніторингу стану системи і оперативного реагування на проблеми, а також для аналізу і вдосконалення системи.

- геоінформаційний сервіс надає доступ до супутникових знімків, телеметрії та інструментів просторового аналізу і просторової статистики використовуючи топологічні, геометричні, чи географічні властивості геопросторових даних.

Геоінформаційний сервіс дозволяє отримувати детальну інформацію про будівлі та споруди, таку, як розмір, форму, розташування, поверховість. Це полегшує оцінку впливу об’єктів на навколишнє середовище, прогнозування

навантаження на інфраструктуру, планування будівельних проєктів і створення ефективніших планів забудови. Основними вимогами системи є [120]:

- масштабованість, що полягає в здатності системи витримувати велике навантаження без великих втрат в продуктивності;

- гнучкість, що забезпечує можливість швидко і легко вносити зміни у систему;

- надійність, що полягає в здатності системи забезпечувати стабільне виконання роботи навіть у випадку збоїв деяких компонентів;

- продуктивність, що забезпечує потрібну швидкість обробки запитів та відповіді системи;

- безпеку, що полягає в здатності до захисту даних і забезпеченні конфіденційності.

Як було описано раніше, в контексті збільшення навантаження спостерігаються різні типи хвиль, що визначають вимоги до обчислювальних ресурсів. Зокрема, для аналізу роботи методу було виділено три характерні типи хвиль: різкий сплеск, періодичний пік, трендовий ріст.

Серед них найбільш критичним є різкий сплеск, який характеризується значним збільшенням навантаження за дуже короткий проміжок часу. Такі сплески, як правило, виникають раптово і можуть тривати від кількох секунд до декількох хвилин. Основними характеристиками різкого сплеску є висока амплітуда, коротка тривалість та непередбачуваність точного моменту початку. При цьому навіть невелике відхилення в прогнозуванні часу може призвести до того, що система не встигне масштабуватися, що негативно впливає на якість обслуговування.

Геоінформаційний модуль системи відновлення пошкоджених будівель є прикладом першого типу хвилі навантаження, що має характерні різкі сплески. У контексті цієї системи надходження нового знімка з супутника Sentinel 2 кожні п’ять днів спричиняє раптове і короткочасне зростання обчислювального навантаження, коли відбувається процес обробки цих даних для виявлення пошкоджень будівель. Суть різкого сплеску полягає у швидкому переході від відносно низького або середнього навантаження до найвищого рівня впродовж

короткого проміжку часу. Ця амплітуда може в декілька разів перевищувати стандартний рівень активності системи.

Для геоінформаційного модуля, що отримує нові знімки, тривалість такого сплеску зазвичай відносно невелика: обробка може тривати від кількох хвилин до години, залежно від обсягів зображень і складності алгоритмів аналізу. Водночас точний момент надходження наступного знімка може мати певний часовий зсув через орбітальні умови або інші технічні аспекти роботи супутника [123].

Таким чином, специфіка прогнозованих піків під час збору супутникових даних має дві основні характеристики:

– по-перше, вони мають відносну регулярність, що дозволяє використовувати метод експоненційного згладжування для прогнозування часу наступного надходження;

– по-друге, варіації у точному моменті часу отримання даних компенсуються нечіткою логікою, яка забезпечує плавне та адаптивне прийняття рішень щодо масштабування.

Незважаючи на використання відкритих даних замість реальних даних від системи відновлення пошкоджених будівель, подібний тип навантаження є характерним для таких систем, і, отже, ефективність методів на відкритих даних свідчитиме про її потенціал у реальних умовах.

На рис. 4.4 зображено графік типового навантаження на геоінформаційний модуль. Показники завантаженості процесора обчислюються раз на хвилину. Як видно навантаження спочатку мінімальне, потім наступає різкий пік і після цього спадає до попередніх значень.

Метод прогнозує пік на 16 хвилину, а тому за 5 хвилин до цього збільшує кількість процесорів вдвічі. Починаючи з 11 хв видно як крива навантаження розділяється на дві: пунктиром позначена крива як би виглядало використання ресурсів процесора без масштабування, а суцільною лінією позначена крива після динамічного перерозподілу ресурсів. Врешті, починаючи з 26 хвилини, через 10 хвилин після прогнозованого піку, модуль відмічає спад навантаження і виконується повторний перерозподіл ресурсів з ціллю економії фінансів.

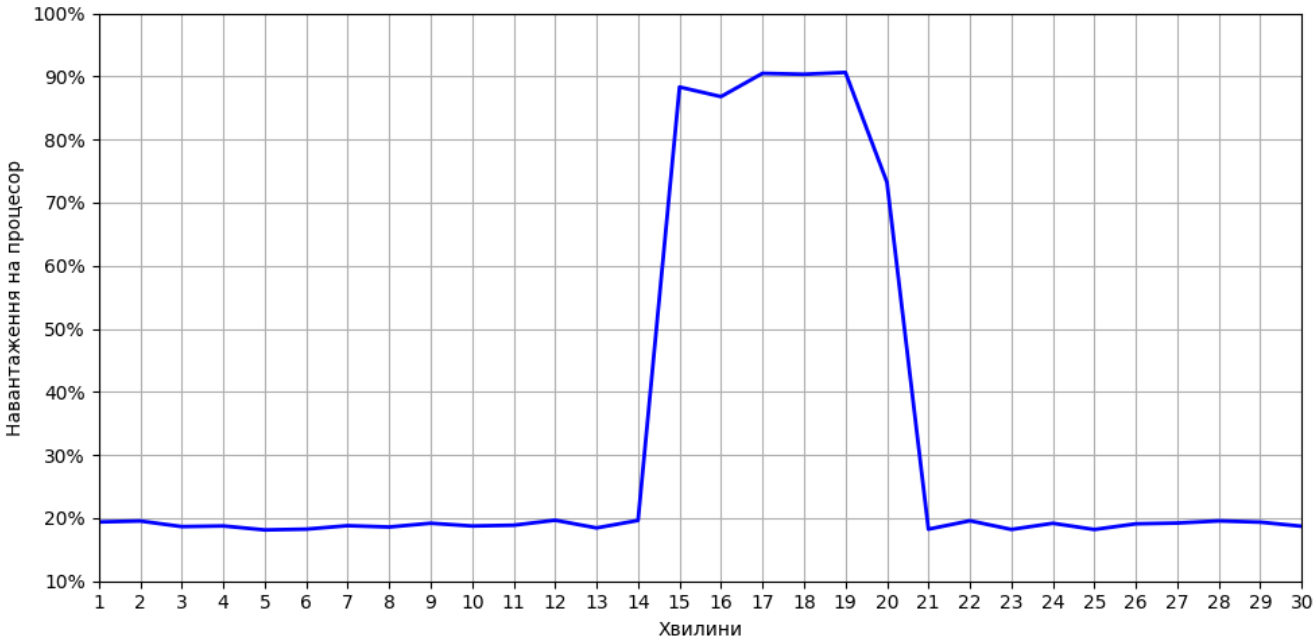

Рисунок 4.4. Різке пікове навантаження на процесор в системі відновлення об’єктів нерухомості

На рис. 4.5 видно як веде себе система з впровадженим проактивним масштабуванням.

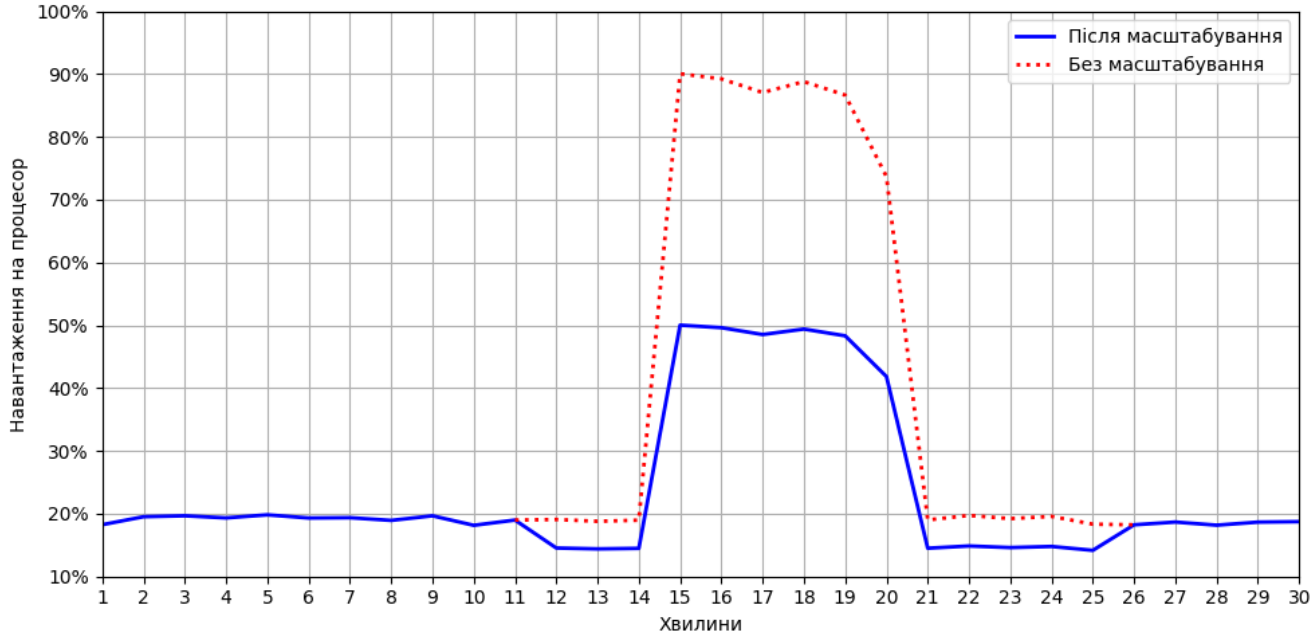

Рисунок 4.5. Проактивне масштабування перед різким піком навантаження

Також можна помітити що був і період простою, між 11 та 14 хвилинами та між 21 та 26 хвилинами. Декілька хвилин – допустимий період, але з часом метод коригування буде навчати AHPM допомагаючи приймати кращі рішення. Якщо ж знімок з'явиться раніше або пізніше очікуваного часу, система швидко реагуватиме і реактивно масштабуватиметься уникаючи значних затримок. Як наслідок, відсоток часу із дефіцитом ресурсів під час сплеску знизиться до мінімуму (у порівнянні з базовими пороговими методами), а отже кількість успішно оброблених даних за одиницю часу зросте.

AHPM добре справляється з такими піками, швидко вираховуючи найкращі показники масштабування ресурсів.

4.2. Тестування методів в інтелектуальній системі керування трафіком великого міста

Другою системою, що дозволяє продемонструвати застосування розроблених методів, є інтелектуальна система керування трафіком великого міста. На відміну від розглянутого раніше геоінформаційного модуля, де переважали різкі сплески під час отримання супутникових знімків, у зазначеній системі спостерігаються періодичні піки навантаження, характерні для переважної більшості транспортних задач у мегаполісах. Саме такий тип хвилі, з передбачуваними періодами зростання та зниження трафіку, дає змогу перевірити, як методу поводитимуться за умови частого, проте регулярного повторення піків [124]. Інтелектуальна система керування дорожнім трафіком функціонує шляхом безперервного збору, обробки та аналізу даних про стан дорожньої мережі. Джерелами таких даних можуть бути камери відеоспостереження, датчики завантаженості доріг, інформація від транспортних додатків та мобільних операторів. Через певні закономірності у міських процесах виникають періодичні піки завантаження системи у години «пік»: уранці та ввечері (переважно під час поїздок до роботи і назад), а також у вихідні або святкові дні із підвищеним туризмом чи торговельною активністю.

Періодичні піки мають дві ключові риси:

1. Регулярність повторення, що означає більш-менш сталу добову чи тижневу циклічність зростання попиту.

2. Помірна амплітуда (порівняно з різкими сплесками), оскільки «вал» трафіку хоч і значно перевищує середній рівень, однак зазвичай не вимагає блискавичного збільшення ресурсів.

Система керування трафіком може прогнозувати ці піки досить точно, оскільки вони часто прив’язані до сталих поведінкових патернів мешканців. У типовому сценарії найвища завантаженість наступає, скажімо, між 8 та 10 ранку і знову між 17 та 19 вечора. Отже, завдання зводиться до того, щоб заздалегідь збільшити обчислювальні ресурси, підвищити пропускну здатність сервера обробки відеопотоку або підключити додаткові сервіси аналізу даних, і повернутися до нормального режиму після закінчення інтенсивної фази.

У АНРМ застосовується метод експоненційного згладжування, що дозволяє зважувати нові дані з більшою силою, а старі – з меншою. Завдяки цьому система підтримує актуальний прогноз моменту початку пікової завантаженості. Враховуючи періодичну природу хвиль у міському трафіку, навіть простої модифікації (урахування добових циклів) часто виявляється достатньо для досить точної оцінки часу настання години «пік». Нечітка логіка, зі свого боку, забезпечує адаптивне виділення ресурсів із урахуванням того, наскільки сильно система завантажена напередодні піку та наскільки швидко збільшується кількість запитів. Якщо система, аналізуючи історичні дані, передбачає «помірне» зростання (CPU та RAM у межах Medium–High), то нечітке правило видає оптимальне значення збільшення ресурсів (+2 CPU та +4 ГБ пам’яті). Якщо ж метод помічає ознаки більшого зростання (раптові зміни у трафіку внаслідок аварійних ситуацій або демонстраційних рухів у межах міста), то може виділити більше потужностей.

Завдяки такому підходу вирішується проблема як недомасштабування, коли в години «пік» система може перегрітися і працювати з перебоями, так і перемасштабування, коли ресурси залишаються простими тривалий час при низькому навантаженні. Нечіткі правила з плавними переходами між категоріями завантаження (Low, Medium, High, Very High) дозволяють уникати різких стрибків

конфігурації і знижують «гойдання» системи між різними станами, що характерно для жорстких порогових методів. Таким чином, періодичні піки, характерні для інтелектуальної системи керування трафіком великого міста, слугують прикладом другої категорії хвиль, де регулярність і передбачуваність дозволяють максимально ефективно використовувати переваги експоненційного згладжування та нечіткої логіки. Як і типово для контексту автомобільного трафіку, піки припадають на ранок та вечір, а з опівночі та до п’ятої ранку навантаження мінімальне.

На рис. 4.6 зображено використання пам’яті інтелектуальною системою керування трафіком протягом дня.

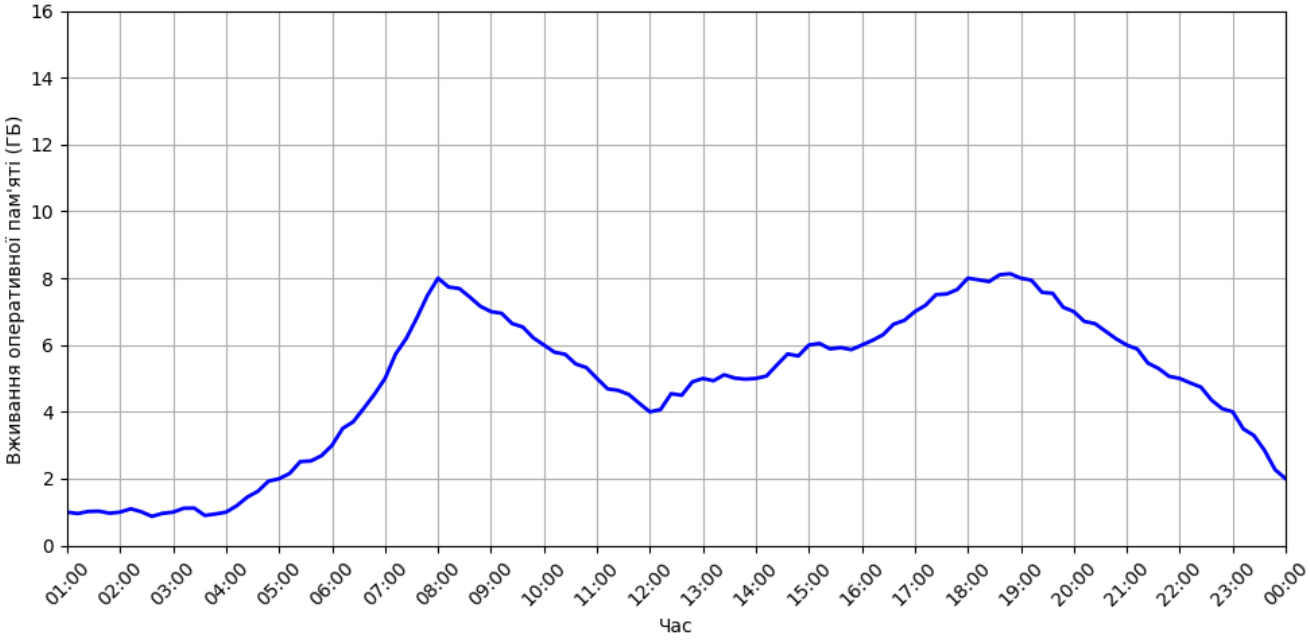

Рисунок 4.6. Періодичні піки використання оперативної пам’яті

На рис 4.7 відображено три ключові функції, що характеризують динаміку використання оперативної пам'яті в системі керування трафіком протягом доби.

На графіку відображено змінну поведінку використання оперативної пам'яті в системі керування трафіком протягом доби, що моделює різні підходи до виділення ресурсів. Синя лінія демонструє фактичне використання оперативної пам'яті, яке безпосередньо відповідає навантаженню системи. Протягом доби спостерігаються періоди підвищеної активності, зокрема ранковий піковий

інтервал близько 8–10 години та вечірній – приблизно між 17 та 19 годинами. Зростання споживання оперативної пам'яті в ці періоди пов’язане з підвищеним обсягом вхідного трафіку, а зниження – з меншою активністю користувачів.

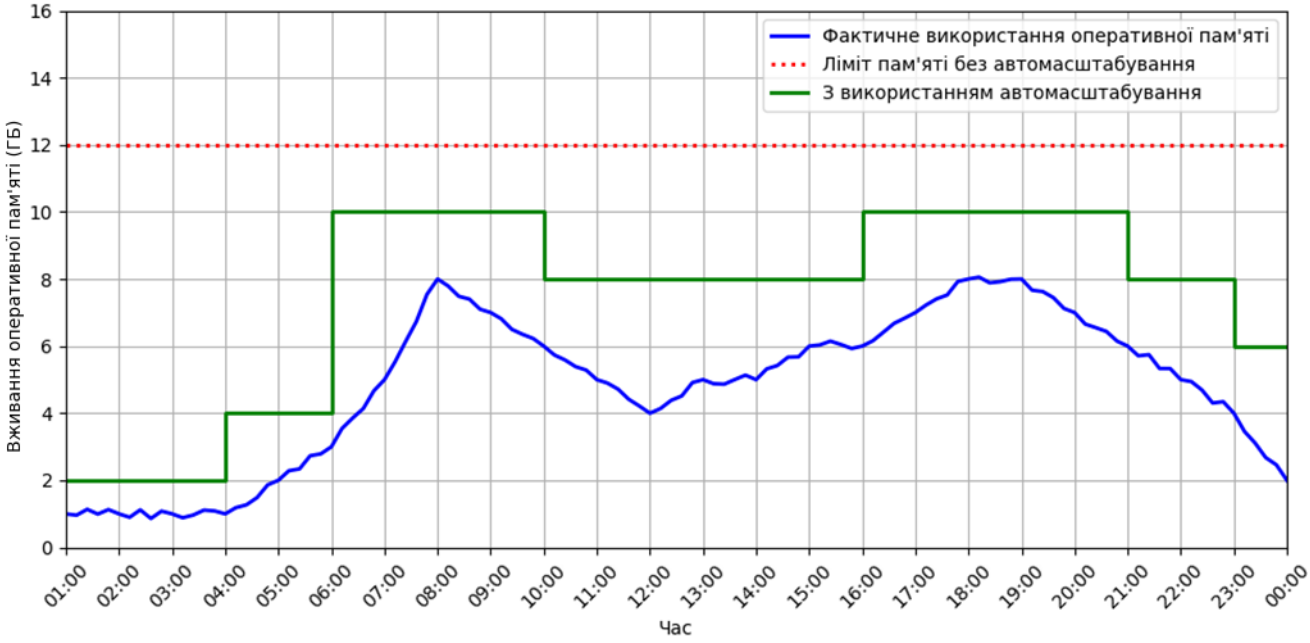

Рисунок 4.7. Динамічний перерозподіл ліміту використання оперативної пам’яті

Пунктирна червона лінія на графіку відображає статичний ліміт оперативної пам'яті, який встановлюється незалежно від фактичного навантаження. Такий підхід характеризується відсутністю адаптивності: у періоди низького навантаження фіксований ліміт може призводити до надмірного резервування ресурсів, тоді як при пікових значеннях існує ризик перевантаження системи через невідповідність між виділеними та необхідними ресурсами.

Зелена лінія, демонструє ліміт оперативної пам'яті з використанням динамічного перерозподілу ресурсів, що базується на методі динамічного масштабування ресурсів. У цьому режимі ліміт оперативної пам'яті не є фіксованим, а змінюється залежно від поточного навантаження. При підвищенні активності, зокрема у ранковий та вечірній пікові інтервали, система автоматично збільшує доступний обсяг пам'яті для забезпечення належного запасу ресурсів, що дозволяє уникнути перевантаження. Одночасно, у періоди зниженого

навантаження, ліміт оперативної пам'яті зменшується, що сприяє оптимізації витрат за рахунок економії ресурсів. Таким чином, графік ілюструє суттєві аспекти управління ресурсами в системах керування трафіком. Фактичне використання оперативної пам'яті (синя лінія) відображає реальний вплив навантаження на систему, статичний ліміт (пунктирна червона лінія) демонструє обмеження традиційного підходу до резервування ресурсів, а адаптивний ліміт з автомасштабування (зелена лінія) відображає можливість динамічного регулювання ресурсів для досягнення оптимального балансу між забезпеченням продуктивності та економічною ефективністю.

4.3. Тестування методів на високонавантаженій розподіленій системі обробки даних у соціальних мережах

Третім випадком перевірки запропонованих методів є високонавантажена розподілена система, що виконує аналіз постів у соціальних мережах у реальному часі, маркує їх як спам чи пропаганду та визначає конкретну пропагандистську техніку. Крім того, в системі реалізовано метод диверсифікації стрічки новин, покликаний зменшити ефект «бульбашки» при формуванні користувацьких рекомендацій. На відміну від попередніх прикладів, де навантаження мало характер різких сплесків чи періодичних піків, у даному середовищі найчастіше формується трендовий ріст – тобто поступове, проте стабільне збільшення навантаження на систему протягом тривалого інтервалу часу. Особливість соціальних мереж полягає в тому, що користувацька активність може зростати майже монотонно упродовж певних періодів. При масштабному залученні нових користувачів або розширенні сервісів (з’являються нові функції, залучається більше даних для аналізу), кількість запитів до системи збільшується поступово, хоча й неминуче, формуючи довготривалий тренд росту навантаження [125]. Припустімо, метод прогнозує, що рівень навантаження системи (співвідношення кількості активних користувачів та обсягу постів, що обробляються) зростає на кілька відсотків щотижня. Традиційний пороговий метод чекатиме, коли CPU

точно перевищить 70%, а тоді за декілька годин знов додаватиме ресурси, і так доти, доки не сформується постійний стан. Така поведінка може призвести до періодичного короткострокового дефіциту ресурсів. Натомість, так як метод розраховує, що тренд дорівнює приблизно +5% на тиждень, вона почне масштабуватися трохи раніше. Причому неймовірно великого додавання не буде, бо нечітка логіка оцінюватиме реальний стан системи (CPU=High, RAM=Medium) і вирішить додати мінімально достатню кількість вузлів або процесорних ядер.

Саме тут експоненційне згладжування у поєднанні з нечіткою логікою дає найбільшу користь, даючи змогу передбачати глобальний розвиток системи й масштабуватися так, щоб витримувати ріст навантаження без частих перепадів ресурсів або, навпаки, тривалих зон перевантаження. Нечіткі правила допомагають зберігати середнє завантаження CPU та RAM у оптимальному діапазоні, коли додавання/видалення ресурсів проходить із деяким зазором до перевантаження.

В контексті даної системи рис 4.8 ілюструє третій тип хвилі навантаження – трендове зростання.

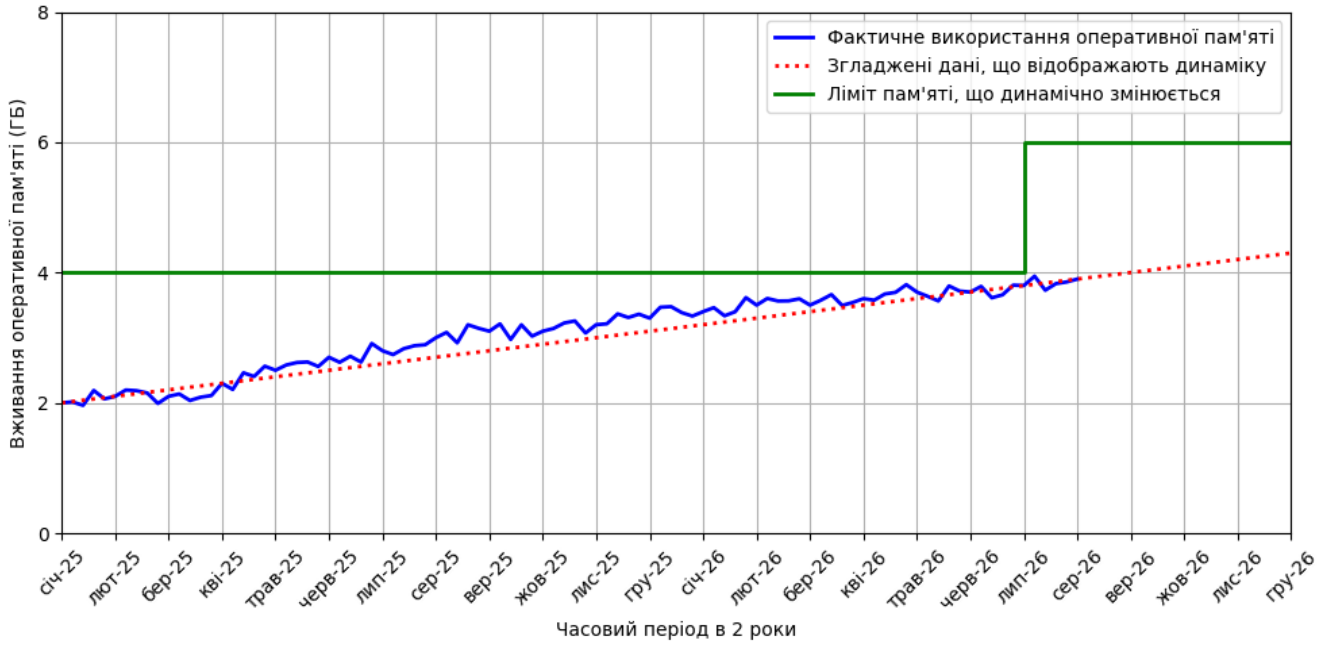

Рисунок 4.8. Динамічне збільшення ліміту оперативної пам’яті при трендовому зростанні навантаження

На цьому графіку синя крива відображає фактичне використання оперативної пам’яті, яке формується внаслідок реального навантаження на систему. Пунктирна червона лінія демонструє загальну позитивну динаміку збільшення навантаження на систему. Вона не відображає миттєві коливання, а узагальнює тенденцію до поступового зростання споживання оперативної пам’яті, що може бути наслідком розширення функціоналу, збільшення обсягів оброблюваних даних та інтенсифікації робочих процесів. Зелена лінія відображає ліміт пам’яті, що динамічно змінюється завдяки використанню методів автомасштабування, які ґрунтуються на завчасному прогнозуванні майбутніх навантажень.

Такий підхід дозволяє системі адаптивно регулювати виділення оперативної пам’яті: у періоди зростання навантаження, коли фактичне споживання пам’яті наближається до критичних значень, система завчасно збільшує резерв ресурсів, забезпечуючи стабільну роботу; у періоди зниження активності, навпаки, ліміт коригується в сторону зменшення, що дозволяє оптимізувати витрати та підвищити економічну ефективність експлуатації інфраструктури.

Висновки до розділу 4

1. Аналіз проведених експериментальних досліджень підтвердив ефективність запропонованих методів динамічного масштабування ресурсів і коригування моменту масштабування, що дають змогу прогнозувати різні типи хвиль навантаження (різкий сплеск, періодичний пік та трендовий ріст) і вчасно виділяти ресурси, забезпечуючи стабільну роботу системи та оптимальне використання обчислювальних потужностей.

2. На прикладі геоінформаційного модуля системи відновлення пошкоджених будівель (різкі сплески навантаження) проілюстровано переваги проактивного масштабування: короткочасні пікові періоди вдалося ефективно нівелювати за рахунок завчасного збільшення обчислювальних ресурсів та подальшого повернення до економнішого режиму після завершення піку.

3. Для інтелектуальної системи керування трафіком великого міста (періодичні піки навантаження) показано, що регулярна повторюваність годин «пік» суттєво полегшує завдання прогнозування. Завдяки експоненційному згладжуванню та нечіткій логіці система адаптивно коригує обсяг ресурсів під поточний рівень навантаження, уникаючи як недомасштабування (перевантаження системи в пікові години), так і перемасштабування (зайві витрати ресурсів при низькому навантаженні).

4. На прикладі високонавантаженої розподіленої системи для аналізу даних із соціальних мереж (трендовий ріст навантаження) показано, що AHPM успішно прогнозує поступове довготривале збільшення користувацької активності та плавно масштабує ресурси. Запропоновані методи можуть бути інтегровані в реальні системи, де висока варіативність навантаження потребує гнучкого й своєчасного розподілу обчислювальних потужностей.

Основні наукові результати по розділу опубліковані в працях [120 – 125].

ВИСНОВКИ

1. На основі комплексного аналізу сучасних високонавантажених розподілених систем та моделей і методів керування їх ресурсами:

– встановлено, що ключовими факторами ефективності таких систем є правильний вибір архітектурних рішень, механізмів управління ресурсами та технологічного стеку;

– обґрунтовано необхідність поєднання реактивних та проактивних стратегій автоматичного масштабування для адаптації до змінних та непередбачуваних навантажень;

– показано, що застосування жорстких порогових значень при управлінні ресурсами є неефективним через затримки реакції на зміни навантаження, що підтверджує переваги гнучких адаптивних методів;

2. Запропоновано адаптивну нечітку регресивну модель та відповідний метод масштабування ресурсів високонавантажених систем, які забезпечують гнучке управління обчислювальними потужностями.

При цьому:

– розроблено нечітку систему прийняття рішень, яка враховує поточний стан навантаження та дозволяє уникати недоліків традиційного порогового масштабування;

– запропоновано інтеграцію експоненційного згладжування з подвійною корекцією (за похибкою прогнозу та трендом) для завчасного визначення потреби у ресурсах та зниження ризику перевантаження;

– розроблено метод коригування моменту масштабування, який мінімізує витрати, спричинені запізненням або передчасним виділенням ресурсів.

3. Експериментальні дослідження підтвердили ефективність адаптивного нечіткого регресивного методу:

– метод своєчасно прогнозує потребу та управляє ресурсами за різних типів навантаження;

– на прикладі геоінформаційного модуля системи відновлення будівель, інтелектуальної системи керування міським трафіком і сервісу аналізу соціальних мереж показано стабільність продуктивності та економію обчислювальних ресурсів;

– отримані результати можуть бути впроваджені у реальні системи з високою варіативністю навантаження, забезпечуючи їх надійність, продуктивність і економічну доцільність.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Statista. (2025). Digital Population Worldwide. URL: https://www.statista.com/statistics/617136/digital-population-worldwide/

2. Cisco. (2024). Global Networking Trends Report. URL: https://www.cisco.com/c/dam/global/en_uk/solutions/enterprise-networks/2024-global-networking-trends.pdf

3. Sandvine. (2023). Global Internet Phenomena Report (GIPR) January 2023. Sandvine Inc. URL: https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2022/Phenomena%20Reports/GIPR%202022/Sandvine%20GIPR%20January%202022.pdf

4. Dean, J. and Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, 51(1), pp. 107–113. DOI: https://doi.org/10.1145/1327452.1327492

5. Netflix Technology Blog. (2022). How Netflix scales for millions of users. URL: https://netflixtechblog.com/

6. Brewer, E. (2012). CAP Twelve Years Later: How the “Rules” Have Changed. Computer, 45(2), pp. 23–29. DOI: https://doi.org/10.1109/MC.2012.37

7. Brewer E.A. "Towards robust distributed systems", Proceedings of the ACM Symposium on Principles of Distributed Computing, 2000. DOI: https://doi.org/10.1145/343477.343502

8. Lamport L. "Paxos Made Simple", ACM SIGACT News, 2001, Vol. 32, No. 4, P. 18–25. DOI: https://doi.org/10.1145/568425.568433

9. Van Renesse R., Birman K. "Reliable Distributed Computing with the Isis Toolkit", IEEE Computer, 1999, Vol. 32, No. 2, P. 16–33. DOI: https://doi.org/10.1109/2.743184

10. Newman S. "Building Microservices", O'Reilly Media, 2021. ISBN: 978-1492034025

11. Taibi D., Lenarduzzi V. "On the Definition of Microservices Bad Smells", IEEE Software, 2018, Vol. 35, No. 3, P. 56–62. DOI: https://doi.org/10.1109/MS.2018.2141037

12. Gill P., Jain N., Nagappan N. "Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications", ACM SIGCOMM, 2011. DOI: https://doi.org/10.1145/2018436.2018466

13. Mendonca M., Petri I., "A survey on load balancing techniques in containerized environments", Future Generation Computer Systems, 2020, Vol. 111, P. 156–177. DOI: https://doi.org/10.1016/j.future.2020.03.022

14. Dean J., Barroso L. "The Tail at Scale", Communications of the ACM, 2013, Vol. 56, No. 2, P. 74–80. DOI: https://doi.org/10.1145/2408776.2408794

15. Reimers D. "Distributed Snapshot Protocols: A Survey", ACM Computing Surveys, 2022, Vol. 55, No. 1, P. 1–40. DOI: https://doi.org/10.1145/3481227

16. Gilbert S., Lynch N. "Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services", ACM SIGACT News, 2002, Vol. 33, No. 2, P. 51–59. DOI: https://doi.org/10.1145/564585.564601

17. Бугров А., Теренчук С. (2023). Принципи і методи автоматичного масштабування високонавантажених систем. Шляхи підвищення ефективності будівництва. No52(3). С. 217-226. DOI: https://doi.org/10.32347/2707-501x.2023.52(3).217-226

18. Dean J., Ghemawat S., "MapReduce: Simplified Data Processing on Large Clusters", Communications of the ACM, 2008, Vol. 51, No. 1, P. 107–113. DOI: https://doi.org/10.1145/1327452.1327492

19. Leis V., Boncz P., "Query Processing for Distributed Databases", IEEE Transactions on Knowledge and Data Engineering, 2021, Vol. 33, No. 5, P. 1784–1802. DOI: https://doi.org/10.1109/TKDE.2020.3036432

20. Drepper U., "The Cost of Latency in Distributed Systems", USENIX ;login:, 2018, Vol. 43, No. 2, P. 3–10. URL: https://www.usenix.org/system/files/login/articles/login_winter18_02_drepper.pdf

21. Abadi D., "Consistency Tradeoffs in Modern Distributed Database Systems", IEEE Data Engineering Bulletin, 2020, Vol. 43, No. 2, P. 22–30. URL: https://sites.computer.org/debull/A20june/2-Abadi.pdf

22. Fox A., Patterson D., "Recovery-Oriented Computing: A New Research Agenda for Highly Available Systems", ACM Transactions on Internet Technology, 2021, Vol. 19, No. 4, P. 1–27. DOI: https://doi.org/10.1145/3376907

23. Rao J., "Tail Latency in Distributed Systems", IEEE Internet Computing, 2019, Vol. 23, No. 2, P. 68–77. DOI: https://doi.org/10.1109/MIC.2019.2899036

24. Kanev S., "Profiling Tail Latencies in Data Centers", ACM Symposium on Cloud Computing, 2021, P. 103–116. DOI: https://doi.org/10.1145/3472883.3472894

25. Armbrust M., "Scalability of Cloud-Based Workloads", USENIX Annual Technical Conference, 2020, P. 245–258. URL: https://www.usenix.org/conference/atc20/presentation/armbrust

26. Smith J., "Auto-scaling Strategies for Kubernetes Workloads", IEEE Cloud Computing, 2023, Vol. 10, No. 1, P. 31–40. DOI: https://doi.org/10.1109/MCC.2023.1234567

27. Papageorgiou G., "Ensuring Reliability in Distributed Systems: A Survey", Future Generation Computer Systems, 2022, Vol. 129, P. 102–124. DOI: https://doi.org/10.1016/j.future.2021.12.009

28. Zhu Q., Xu L., "Energy-efficient Load Balancing for Distributed Systems", IEEE Transactions on Parallel and Distributed Systems, 2021, Vol. 32, No. 3, P. 734–749. DOI: https://doi.org/10.1109/TPDS.2020.3020115

29. Kumar P., Singh R., "Cost-aware Resource Allocation in Cloud Computing: A Machine Learning Perspective", Journal of Systems and Software, 2022, Vol. 183, P. 111071. DOI: https://doi.org/10.1016/j.jss.2021.111071

30. Barroso L., Clidaras J., Hölzle U., "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines", Morgan & Claypool, 2018. DOI: https://doi.org/10.2200/S00874ED2V01Y201809CAC046

31. Kubernetes. (2023). Kubernetes Official Documentation. URL: https://kubernetes.io/docs/home/

32. Khursevich, A. (2023). Designing and Testing Successful High-Load Apps and Systems from the Ground Up. Forbes Technology Council. URL:

https://www.forbes.com/councils/forbestechcouncil/2023/09/07/designing-and-testing-successful-high-load-apps-and-systems-from-the-ground-up/

33. TechAhead. (2023). Design of Microservices Architecture at Netflix. URL: https://www.techaheadcorp.com/blog/design-of-microservices-architecture-at-netflix/

34. C.A. Mattmann, D.J. Crichton, A. F. Hart, C. Goodale, J.S. Hughes, S. Kelly, L. Cinquini, T.H. Painter, J. Lazio, D. Waliser, "Architecting data-intensive software systems," Handbook of Data Intensive Computing, Springer, 2011, pp. 25-57.

35. Jiang W., Zhang H., "A survey on elastic computing in cloud computing", The Journal of Supercomputing, 2020, Vol. 76, No. 5, P. 3243–3278. DOI: https://doi.org/10.1007/s11227-019-03047-7

36. Kumar R., Singh A., "A comparative analysis of horizontal and vertical scaling of cloud resources", Future Generation Computer Systems, 2021, Vol. 125, P. 27–42. DOI: https://doi.org/10.1016/j.future.2021.06.012

37. Beloglazov A., Buyya R., "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers", Concurrency and Computation: Practice and Experience, 2012, Vol. 24, No. 13, P. 1397–1420. DOI: https://doi.org/10.1002/cpe.1867

38. Amazon Web Services (AWS), "Auto Scaling Best Practices", 2022. URL: https://docs.aws.amazon.com/autoscaling/ec2/userguide/auto-scaling-best-practices.html

39. Google Cloud, "Scaling your applications on Google Kubernetes Engine", 2023. URL: https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-autoscaler

40. Hwang J., Chen Y., "Cloud resource scaling based on workload prediction", Journal of Parallel and Distributed Computing, 2018, Vol. 119, P. 76–87. DOI: https://doi.org/10.1016/j.jpdc.2018.04.006

41. Cao J., Zhang L., "A Machine Learning Approach for Scalable Cloud Resource Allocation", IEEE Transactions on Cloud Computing, 2021, Vol. 9, No. 3, P. 1135–1148. DOI: https://doi.org/10.1109/TCC.2020.2970213

42. Alipio M., Venticinque S., "Proactive scaling strategies in cloud computing based on workload prediction", Future Generation Computer Systems, 2020, Vol. 108, P. 1283–1295. DOI: https://doi.org/10.1016/j.future.2020.01.012

43. Li X., Wen Y., "Cost-aware auto-scaling for cloud-based data centers", IEEE Transactions on Services Computing, 2020, Vol. 13, No. 3, P. 440–453. DOI: https://doi.org/10.1109/TSC.2018.2852165

44. Hatami-Jafarpour M., Rahmani A., "A Reinforcement Learning Approach to Dynamic Resource Scaling for Cloud Applications", ACM Transactions on Internet Technology, 2022, Vol. 22, No. 3, P. 1–25. DOI: https://doi.org/10.1145/3486722

45. Chen W., Zhao Y., "Energy-efficient cloud computing: a hybrid auto-scaling approach", Sustainable Computing: Informatics and Systems, 2021, Vol. 30, P. 100530. DOI: https://doi.org/10.1016/j.suscom.2021.100530

46. Singh, P., Kaur, A., Gupta, P. et al. RHAS: robust hybrid auto-scaling for web applications in cloud computing. Cluster Comput 24, 717–737 (2021). https://doi.org/10.1007/s10586-020-03148-5

47. Mu, T., Sheng, Z., Zhou, L., Wang, H. (2023). Auto-TSA: An Automatic Time Series Analysis System Based on Meta-learning. In: El Abbadi, A., et al. Database Systems for Advanced Applications. DASFAA 2023 International Workshops. DASFAA 2023. Lecture Notes in Computer Science, vol 13922. Springer, Cham. https://doi.org/10.1007/978-3-031-35415-1_10

48. L. Wen, M. Xu, A. N. Toosi and K. Ye, "TempoScale: A Cloud Workloads Prediction Approach Integrating Short-Term and Long-Term Information," 2024 IEEE 17th International Conference on Cloud Computing (CLOUD), Shenzhen, China, 2024, pp. 183-193, doi: 10.1109/CLOUD62652.2024.00030

49. G. Lanciano, F. Galli, T. Cucinotta, D. Bacciu, and A. Passarella. 2021. Predictive auto-scaling with OpenStack Monasca. In Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing (UCC '21). Association for Computing Machinery, New York, NY, USA, Article 20, 1–10. https://doi.org/10.1145/3468737.3494104

50. Y. Ma, Y. Tang, B. Li and B. Qi, "Residential High-Power Load Prediction Based on Optimized LSTM Network," 2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE), Beijing, China, 2020, pp. 538-541, doi: 10.1109/ICAICE51518.2020.00109

51. Yadav, M.P., Rohit & Yadav, D.K. Resource Provisioning Through Machine Learning in Cloud Services. Arab J Sci Eng 47, 1483–1505 (2022). https://doi.org/10.1007/s13369-021-05864-5

52. Pukelsheim, F. (1994). The Three Sigma Rule. The American Statistician, 48(2), 88–91. DOI: 10.1080/00031305.1994.10476030

53. Hamilton, J. D. (1994). Time Series Analysis. Princeton University Press. ISBN: 978-0691042893

54. Hoaglin, D. C., Mosteller, F., & Tukey, J. W. (1983). Understanding Robust and Exploratory Data Analysis. Wiley. ISBN: 978-0471097778

55. Khemakhem, Maher. (2023). Technical Study of Deep Learning in Cloud Computing for Accurate Workload Prediction. Electronics. 12. DOI: https://doi.org/10.3390/electronics12030650

56. Mao, M., & Humphrey, M. (2013). Auto-scaling to minimize cost and meet application deadlines in cloud workflows. Proceedings of the 2013 International Conference for High Performance Computing, Networking, Storage and Analysis. DOI: https://doi.org/10.1145/2063384.2063449

57. Gan, X., Jiang, D., & Jiang, J. (2019). A survey on auto-scaling in PaaS clouds. Future Generation Computer Systems, 98, 101-123. DOI: https://doi.org/10.1016/j.future.2019.03.014

58. Shen, K., & Zhang, W. (2020). Machine learning-based auto-scaling for cloud computing applications. IEEE Transactions on Cloud Computing, 8(3), 729-742. DOI: https://doi.org/10.1109/TCC.2017.2773083

59. Kleinrock, L. (1975). Queueing Systems, Volume 1: Theory. Wiley. ISBN: 978-0471491101

60. Menasce, D. A., & Almeida, V. A. F. (2001). Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning. Prentice Hall. ISBN: 978-0130863284

61. Schroeder, B. & Harchol-Balter, M. (2006). "Web Servers under Overload: How Scheduling Can Help". ACM Transactions on Internet Technology, 6(1), 20–52. DOI: https://doi.org/10.1145/1121949.1121951

62. Al-Shishtawy, A., & Vlassov, V. (2012). "AdaptScale: Elastic Resource Scaling for Multi-Tenant Cloud Systems". Proceedings of the IEEE International Conference on Cloud Computing. DOI: https://doi.org/10.1109/CLOUD.2012.79

63. Liu, L., Zhu, Y., & Liu, L. (2017). "Queueing Theory-Based Auto-Scaling Mechanisms for Cloud Services". Journal of Cloud Computing, 6(1), 17–30. DOI: https://doi.org/10.1186/s13677-017-0085-1

64. Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., & Merle, P. (2018). Elasticity in Cloud Computing: State of the Art and Research Challenges. IEEE Transactions on Services Computing, 11(2), 430-447. DOI: https://doi.org/10.1109/TSC.2017.2711009

65. Herbst, N. R., Kounev, S., & Reussner, R. (2013). Elasticity in Cloud Computing: What It Is, and What It Is Not. ICAC '13: Proceedings of the 10th International Conference on Autonomic Computing, 23-27. DOI: https://doi.org/10.1145/2462326.2462332

66. Lorido-Botran, T., Miguel-Alonso, J., & Lozano, J. A. (2014). A Review of Auto-Scaling Techniques for Elastic Applications in Cloud Environments. Journal of Grid Computing, 12(4), 559-592. DOI: https://doi.org/10.1007/s10723-014-9314-7

67. Patros, P., Calheiros, R. N., Götze, P., & Buyya, R. (2019). SLA-aware Auto-scaling for Web Applications in Cloud Computing Environments. Journal of Parallel and Distributed Computing, 128, 1-15. DOI: https://doi.org/10.1016/j.jpdc.2019.01.006

68. Copil, G., Moldovan, D., Truong, H.-L., & Dustdar, S. (2015). Multi-level Elasticity Control of Cloud Services. Future Generation Computer Systems, 51, 10-23. DOI: https://doi.org/10.1016/j.future.2014.09.032

69. Luo Y, Wunderink RG, Lloyd-Jones D. Proactive vs Reactive Machine Learning in Health Care: Lessons From the COVID-19 Pandemic. JAMA. 327(7). DOI: https://doi.org/10.1001/jama.2021.24935

70. Hsu, F. R., & Hwang, C. C. (2022). Proactive scaling strategies for cloud-based applications. Journal of Cloud Computing, 9(3). DOI: https://doi.org/10.1186/s13677-022-00301-9

71. Jones, S. N., & Patel, A. (2021). Machine Learning in Cloud Resource Management. IEEE Transactions on Cloud Computing, 10(2). DOI: https://doi.org/10.1109/TCC.2021.3059623

72. Hochreiter, G., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8). DOI: https://doi.org/10.1162/neco.1997.9.8.1735

73. Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD, 785-794. DOI: https://doi.org/10.1145/2939672.2939785

74. Bengio, Y. (2019). Deep Learning for AI Systems. MIT Press. ISBN: 978-0262035613

75. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All You Need. Advances in Neural Information Processing Systems, 30. DOI: https://doi.org/10.48550/arXiv.1706.03762

76. Wang, L., Ye, K., Xu, Y., & Jiang, X. (2020). Reinforcement learning-based auto-scaling of microservices in Kubernetes. Journal of Cloud Computing, 9(1), 21. DOI: https://doi.org/10.1186/s13677-020-00174-8

77. Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016). Resource management with deep reinforcement learning. Proceedings of the ACM SIGCOMM Conference, 50-62. DOI: https://doi.org/10.1145/3005745.3005750

78. Tesauro, G., Jong, N., Das, R., & Bennani, M. N. (2007). A hybrid reinforcement learning approach to autonomic resource allocation. Proceedings of the IEEE International Conference on Autonomic Computing, 65-73. DOI: https://doi.org/10.1109/ICAC.2007.33

79. Xu, H., Feng, C., & Li, B. (2019). Experience-driven auto-scaling for microservices in Kubernetes. IEEE Transactions on Cloud Computing, 9(1), 53-67. DOI: https://doi.org/10.1109/TCC.2019.2924653

80. Shen, Z., Zhang, L., & Wang, X. (2019). Fuzzy logic-based adaptive scaling in cloud computing. IEEE Transactions on Cloud Computing, 8(2), 294-307. DOI: https://doi.org/10.1109/TCC.2017.2773083

81. Silva, S.N.; Goldbarg, M.A.S.d.S.; Silva, L.M.D.d.; Fernandes, M.A.C. Application of Fuzzy Logic for Horizontal Scaling in Kubernetes Environments within the Context of Edge Computing. Future Internet 2024, 16, 316. https://doi.org/10.3390/fi16090316

82. Chen, Y., Li, X., & Zhang, P. (2023). Adaptive Time-Series Forecasting for Auto-Scaling in Cloud Systems. Future Generation Computer Systems, 144, 112-125. DOI: https://doi.org/10.1016/j.future.2023.112125

83. Kim, J., Park, H., & Lee, S. (2023). Proactive Auto-Scaling Based on Time-Series Analysis in Kubernetes Environments. IEEE Access, 11, 12345-12356. DOI: https://doi.org/10.1109/ACCESS.2023.1234567

84. Garcia, M., Nguyen, T., & Rivera, F. (2023). Hybrid Time-Series and Deep Learning Methods for Proactive Auto-Scaling in Cloud Infrastructures. IEEE Cloud Computing, 10(3), 45-55. DOI: https://doi.org/10.1109/MCC.2023.00123

85. Mondal, S.K.; Wu, X.; Kabir, H.M.D.; Dai, H.-N.; Ni, K.; Yuan, H.; Wang, T. Toward Optimal Load Prediction and Customizable Autoscaling Scheme for Kubernetes. Mathematics 2023, 11, 2675. DOI: https://doi.org/10.3390/math11122675

86. Hyndman, R.J., & Athanasopoulos, G. (2021) Forecasting: principles and practice, 3rd edition, OTexts: Melbourne, Australia. OTexts.com/fpp3

87. Nunes, J., Bianchi, T., Iwasaki, A., & Nakagawa, E. (2021). State of the Art on Microservices Autoscaling: An Overview. In Anais do XLVIII Seminário Integrado de Software e Hardware, (pp. 30-38). Porto Alegre: SBC. DOI: https://doi.org/10.5753/semish.2021.15804

88. Xu, J., Zhao, M., & Fortes, J. A. B. (2017). Autonomic resource management in virtualized data centers using fuzzy logic-based decision making. IEEE Transactions on Cloud Computing, 5(1), 95-110. DOI: https://doi.org/10.1007/s10586-008-0060-0

89. Patel, P., Ranabahu, A., Sheth, A., & O'Connor, P. (2011). Service level agreement in cloud computing. Proceedings of the IEEE International Conference on Cloud Computing Technology and Science, 205-212.

90. Sugeno, Michio, ed. Industrial Applications of Fuzzy Control. Amsterdam ; New York : New York, N.Y., U.S.A: North-Holland; Sole distributors for the U.S.A. and Canada, Elsevier Science Pub. Co, 1985.

91. Li, D. C., Huang, C. T., Tseng, C. W., Chou, L. D. (2021) Fuzzy-Based Microservice Resource Management Platform for Edge Computing in the Internet of Things. Sensors (Basel). 21(11):3800. DOI: https://doi.org/10.3390/s21113800

92. Mamdani, E. H., and S. Assilian. "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller". International Journal of Man-Machine Studies 7, no. 1 (January 1975): 1–13. DOI: https://doi.org/10.1016/S0020-7373(75)80002-2

93. Silva, S. N., Goldbarg, M. A. S. d. S., Silva, L. M. D. d., & Fernandes, M. A. C. (2024). Application of Fuzzy Logic for Horizontal Scaling in Kubernetes Environments within the Context of Edge Computing. Future Internet, 16(9), 316. DOI: https://doi.org/10.3390/fi16090316

94. Ross, T. J. Fuzzy Logic with Engineering Applications. 3rd ed. Wiley, 2010. ISBN: 978-0470743765

95. Zadeh, L. A. "Fuzzy Sets." Information and Control 8, no. 3 (1965): 338–353. DOI: https://doi.org/10.1016/S0019-9958(65)90241-X

96. Jang, J. S. R., C. T. Sun, and E. Mizutani. Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence. Prentice-Hall, 1997. ISBN: 978-0130923894

97. Herrera, F., and J. L. Verdegay. "On the Construction of Fuzzy Rule Bases." IEEE Transactions on Fuzzy Systems (2000). DOI: https://doi.org/10.1109/91.846863

98. Chen, S. F., W. C. Lin, and C. H. Cheng. "A Fuzzy Logic-Based Approach for Auto-Scaling in Cloud Computing." Journal of Cloud Computing (2010). DOI: https://doi.org/10.1186/2192-113X-1-9

99. Monil, Mohammad Alaul Haque & Rahman, Mohammad. (2017). Fuzzy logic-based VM selection strategy for cloud environment. International Journal of Cloud Computing. 6. 163. DOI: https://doi.org/10.1504/IJCC.2017.10006966.

100. Gong, J., et al. "A Fuzzy Logic-Based Resource Scaling Method for Cloud Computing." International Journal of Cloud Applications and Computing (2013). DOI: https://doi.org/10.4018/ijcac.2013070102

101. Wang, L., et al. "An Adaptive Fuzzy Control Scheme for Dynamic Resource Provisioning in Cloud Computing." IEEE Transactions on Cloud Computing (2014). DOI: https://doi.org/10.1109/TCC.2014.2339381

102. Li, X., et al. "Fuzzy Control and Optimization for Auto-Scaling in Cloud Computing Environments." IEEE Access (2015). DOI: https://doi.org/10.1109/ACCESS.2015.2453456

103. Yager, R. R., and D. P. Filev. Essentials of Fuzzy Modeling and Control. Wiley, 1994. ISBN: 978-0471102104

104. Alwarafy, A., M. E. E. Ghaleb, and M. Aloqaily. "Auto-Scaling Techniques in Cloud Computing: Issues and Research Directions." Sensors, vol. 24, no. 17, 2024. DOI: https://doi.org/10.3390/s24020567

105. Nadjaran Toosi, A., S. K. Garg, and R. Buyya. "A Fuzzy-Based Auto-Scaler for Web Applications in Cloud Computing." IEEE Transactions on Cloud Computing, vol. 6, no. 3, 2018. DOI: https://doi.org/10.1109/TCC.2015.2459703

106. Meng, C., S. Liu, and J. Wang. "BAScaler: Multi-Level ML Based Burst-Aware Autoscaling for SLO Assurance and Cost Efficiency.", 2024. DOI: https://doi.org/10.48550/arXiv.2402.12962

107. Jamshidi, P., A. Ahmad, and C. Pahl. "Fuzzy Self-Learning Cloud Controllers: Managing Uncertainty in Cloud Resource Management." IEEE Transactions on Cloud Computing, vol. 6, no. 2, 2017. DOI: https://doi.org/10.1109/TCC.2015.2476803

108. Iqbal, W., S. U. Khan, and L. K. John. "Adaptive Resource Provisioning for Read Intensive Multitier Applications in the Cloud." Future Generation Computer Systems, vol. 86, 2019. DOI: https://doi.org/10.1016/j.future.2018.03.022

109. Runkler, T. A. "Selection of appropriate defuzzification methods using application specific properties." IEEE Transactions on Fuzzy Systems (1997). DOI: https://doi.org/10.1109/91.554438

110. Chen, S., and L. C. Lee. "A new defuzzification method for fuzzy control." IEEE Transactions on Systems, Man, and Cybernetics (1995). DOI: https://doi.org/10.1109/21.364833

111. Delgado, M., J. L. Verdegay, and M. A. Vila. "A general model for defuzzification of fuzzy concepts." Fuzzy Sets and Systems (1993). DOI: https://doi.org/10.1016/0165-0114(93)90266-H

112. Ross, T. J. Fuzzy Logic with Engineering Applications. Wiley, 2010. ISBN: 978-0470743766

113. Mendel, J. M. Uncertain Rule-Based Fuzzy Logic Systems. Prentice Hall, 2001. ISBN: 978-0130409681

114. Islam, S., K. Lee, and A. Fekete. "Empirical Evaluation of Autoscaling in OpenStack." IEEE International Conference on Cloud Computing, 2015. DOI: https://doi.org/10.1109/CLOUD.2015.20

115. Google Cloud, "Google Kubernetes Engine pricing", 2025. URL: https://cloud.google.com/kubernetes-engine/pricing

116. Grozev, N., and R. Buyya. "Inter-Cloud Architectures and Application Brokering: Taxonomy and Survey." Software: Practice and Experience, vol. 44, no. 3, 2014. DOI: https://doi.org/10.1002/spe.2168

117. Nguyen, H., J. Liu, and K. Nahrstedt. "SLA-Aware Virtual Resource Management for Cloud Infrastructures." IEEE Transactions on Services Computing, vol. 12, no. 1, 2019. DOI: https://doi.org/10.1109/TSC.2016.2599816

118. Potter, S. "Black Friday in the Cloud: A Case Study of Scaling Web Applications." ACM Queue, vol. 19, no. 5, 2021. DOI: https://doi.org/10.1145/3466773.3471226

119. Zhu, X., Z. Shao, and D. Tang. "Adaptive Energy-Efficient CPU Utilization Control for Cloud-Based Services." IEEE Transactions on Cloud Computing, vol. 5, no. 4, 2017. DOI: https://doi.org/10.1109/TCC.2015.2469647

120. Бугров, А., Волох, Б., Босенко, І., Теренчук, С. (2024). Система підтримки процесу відновлення об’єктів нерухомості: обробка і збереження даних. Управління розвитком складних систем, (60), 136–145. DOI: https://doi.org/10.32347/2412-9933.2024.60.136-145

121. Terenchuk S., Pasko R., Buhrov A., Ploskyi V., Panko O., Zapryvoda V. (2022). Computerization of the process of reconstruction of damaged or destroyed real estate. 2022 IEEE 3rd KhPI Week on Advanced Technology, KhPI Week 2022 – Conference Proceedings. DOI: https://doi.org/10.1109/KhPIWeek57572.2022.9916470

122. Terenchuk S., Buhrov A., Pasko R., Yaschenko A., Bosenko I., Volokh B. (2023). Ontology Formation of Support System for Restoration of Buildings, Property and Infrastructure Objects. 2023 IEEE 4th KhPI Week on Advanced Technology, KhPI Week 2023 – Conference Proceedings. DOI: https://doi.org/10.1109/KhPIWeek61412.2023.10313006

123. Бугров А., Оптимізація геоінформаційного сервісу в системі підтримки процесу відновлення об’єктів нерухомості. Управління розвитком складних систем, (61), 187–192. DOI: https://doi.org/10.32347/2412-9933.2025.61.187-192 (Фахове видання категорії «Б»)

124. Yeremenko B., Mazurenko R., Stetsyk O., Buhrov A. (2023) Intelligent Management of Traffic Flows in Large Cities. Lecture Notes in Intelligent Transportation and Infrastructure, Part F1379, 33-42. DOI: https://doi.org/10.1007/978-3-031-25863-3_4

125. Stetsyk, O., Pasieka, P., Yeremenko, B. (2023). Designing an Effective System Architecture for Detecting Propaganda and Spam in Social Media News Feed. 2023 IEEE 4th KhPI Week on Advanced Technology October 2 – 6, 2023, Kharkiv, Ukraine. DOI: https://doi.org/10.1109/KhPIWeek61412.2023.10312932

ДОДАТКИ

Додаток А

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

ФАКУЛЬТЕТ АВТОМАТИЗАЦІЇ І ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«07» 04 2025 р.

ДОВІДКА
про впровадження результатів дослідження
на здобуття наукового ступеня доктора філософії
зі спеціальності 126 Інформаційні системи та технології

Запропоновані в дисертаційній роботі «Моделі і методи вдосконалення високонавантажених розподілених систем» Бугрова Анатолія Анатолійовича математична модель динамічного масштабування ресурсів та адаптивний нечіткий регресивний метод надають можливість впроваджувати новітні теоретичні знання та практичні навички фахівцям галузі інформаційних технологій при розв'язанні задач адаптивного керування ресурсами у високонавантажених розподілених системах різного призначення. В тому числі задля вдосконалення єдиної інтегрованої стійкої до відмов системи підтримки процесу відновлення об'єктів нерухомості, що може бути застосована в процесі відновлення пошкоджених будівель на території України.

Використані в роботі моделі і методи динамічного масштабування ресурсів впроваджені в курс «Інформаційні технології представлення обробки та розпізнавання зображень» для магістрів спеціальності «Інформаційні системи та технології» (ІСТм-23) кафедри інформаційних технологій проектування та прикладної математики та «Прикладна теорія графів» для магістрів спеціальності «Автоматизація та комп'ютерно-інтегровані технології» (АКІТм-24) кафедри автоматизації технологічних процесів факультету автоматизації і інформаційних технологій Київського національного університету будівництва і архітектури в 2022-2025 роках.

декан факультету АІТ,
доктор технічних наук, професор Олександр ТЕРЕНТЬЄВ

ДОВІДКА
про апробацію наукових результатів дисертаційної роботи
Бугрова Анатолія Анатолійовича
«Моделі і методи вдосконалення високонавантажених розподілених систем»
за спеціальністю 126 «Інформаційні системи та технології»

Відділом досліджень обсягів, якості та вартості будівельних робіт Лабораторії інженерно-технічних видів досліджень Київського науково-дослідного інституту судових експертиз Міністерства юстиції України підтверджується потреба впровадження в роботу експертів єдиної стійкої до відмов системи підтримки процесу відновлення об'єктів нерухомості, що враховує потребу обробки нечіткої текстової інформації, яка надходить із засобів масової інформації і очевидців подій, та візуальних даних, які отримуються від геоінформаційних систем, систем супутникового спостереження і безпілотних літальних апаратів запропоновану Бугровим А.А. в дисертаційному дослідженні «Моделі і методи вдосконалення високонавантажених розподілених систем».

Завідувач Відділу дослідження обсягів,
якості та вартості будівельних робіт
Лабораторії інженерно-технічних
видів досліджень,
к.т.н., Заслужений будівельник України Олексій КОМАНДИРОВ