УДК 004.75 http://doi.org/10.15407/pp2024.02-03.215

*О.А. Дмитренко, М.А. Скулиш*

# ВИЗНАЧЕННЯ МІКРОПРОЦЕСОРНИХ ГРУП ДЛЯ ЕФЕКТИВНОГО ВИКОРИСТАННЯ ПРОЦЕСОРНИХ ПОТУЖНОСТЕЙ

Задача оптимізації використання ресурсів хмарних систем надзвичайно актуальна, оскільки ця технологія стала дуже поширеною та легко доступною [1]. Розробники хмарних систем постійно вдосконалюють їх, створюючи нові та ефективні сервіси для швидкого розгортання застосунків. Серед таких сервісів можна виділити серверні технології, власні бази даних та інші інструменти. Тестування продуктивності додатків у хмарі стало значно простішим завдяки появі Docker та Kubernetes, що привело до ще більшого попиту на хмарні ресурси. Оскільки на розробку та підтримку інфраструктури хмар витрачається величезна кількість ресурсів, важливо розуміти, яким чином ці ресурси можна оптимізувати.

Стаття розглядає ключові характеристики комп'ютерних систем, які можна виміряти та впливати на них. Ці характеристики включають пропускну здатність каналів передачі, оцінку ефективності роботи за затримками, обсяг оперативної та постійної пам'яті, потужність обробки та кількість ядер.

Пропонується алгоритм ідентифікації доповнювальних екземплярів мікросервісів, які могли б ефективно використовувати ресурси сервера. Спочатку екземпляри мікросервісів класифікуються за їхньою потужністю, розглядаючи малі екземпляри як одиницю потужності. Шляхом аналізу екземпляри мікросервісів групуються у класи еквівалентності за схожістю. Екземпляри потім сортуються за амплітудою використання ресурсу. Ідеально, якщо екземпляри зі значними відмінностями у навантаженні об'єднуються з іншими схожими за амплітудою, але протилежними за фазою, щоб максимізувати використання ресурсів. Комбінації можуть бути доречними також з екземплярами з низькою амплітудою. В межах протилежних класів, що відрізняються фазами активності екземплярів мікропроцесорів, алгоритм шукає перший екземпляр мікросервісу, який відповідає умовам. Для цього зберігається статистика комбінацій з усіма екземплярами до знаходження першої успішної комбінації. В протилежному випадку відбувається пошук найменш невдалої комбінації.

Ключові слова: хмарні системи, мікросервісна архітектура, ефективність процесора, оптимізація енергоспоживання, зниження вартості технології

*O. Dmytrenko, M. Skulysh*

# PPROCESSOR GROUP DETERMINATION FOR THE EFFECTIVE PROCESSOR CAPACITY USAGE

The optimization of resource utilization in cloud systems is a critical endeavor given the widespread adoption of cloud technology and its user-friendly nature [1]. Cloud system developers are continuously innovating to deliver fast application performance, leveraging advancements such as serverless architectures and proprietary databases. The introduction of Docker and Kubernetes has further facilitated performance testing in the cloud, resulting in a significant uptick in cloud usage. Considering the substantial investments made in cloud technology, it is imperative to explore strategies for optimizing resource allocation and utilization

This article delves into the essential metrics of computer performance that can be both measured and influenced. These metrics include channel capacity, latency assessment, memory types (operational and non-volatile), processing power, and core count. Understanding and effectively managing these metrics are crucial for maximizing the efficiency of cloud systems.

The proposed algorithm outlined in the article aims to identify complementary instances of microservices that can efficiently share server resources. Initially, microservices are categorized into instances with similar capacity levels, forming equivalence classes based on their resource usage patterns. Within these classes, instances are further sorted based on their resource utilization amplitudes. The goal is to pair instances with significant differences in resource utilization with others exhibiting similar amplitudes to optimize resource allocation. Additionally, instances with lower resource utilization may also be combined to maximize resource efficiency. The algorithm iteratively searches for compatible microservice combinations within these equivalence classes until suitable matches are found. Throughout the process, statistics of attempted combinations are maintained to inform future optimization strategies.

Key words: cloud systems, microservices architecture, processor efficiency, energy efficiency optimization, technology cost reduction

## Вступ

Використання хмарних систем почало поширюватись на початку 2000-х років. Водночас розробляти та тестувати застосунки під хмару було незручно до появи Docker у 2013 році [1]. Система контейнеризації, а до неї ще й система оркестрації Kubernetes, дали новий поштовх мікросервісній архітектурі, яка і без того набирала оберти, адже стало можливо імітувати роботу та розростання мікросервісів на хмарі.

Мікросервісна архітектура своєю чергою виправдана для великих застосунків, які містять окремий функціонал, що не завжди міцно зв'язаний з іншими підсистемами [2]. Користувачі застосунків можуть проводити багато часу, роздивляючись сайт чи обираючи щось, та мало часу використовувати на інші дії. Мікросервіси дають можливість розділити ці частини, і, в разі сильного навантаження, запустити додаткові інстанси (сервери) тільки з тими частинами застосунку, які цього потребують.

Така методика вже економить як гроші власників бізнесу, адже під час запуску додаткового застосунку та розподілення навантаження між ними (load-balancing) витрачатиметься більше ресурсів, аніж для запуску окремого елемента [1].

Для забезпечення стабільної роботи всіх елементів хмарної системи оновлення серверів зазвичай робиться, коли їхній гарантійний термін збіг. Таким чином найкраще максимально скористатись серверами, використавши їхній ресурс на повну.

В цій статті описується алгоритм, що допоміг би хмарній системі не тільки ефективно розподіляти навантаження між мікросервісами та оркеструвати запуск чи загасання додаткових, а ще і на повну використовувати серверні та канальні потужності.

У процесі роботи застосунку на сервері зазвичай сервер завантажений не на повну. Ступінь завантаження залежить від призначення застосунку, часу доби та дня тижня. Навантаження - це прогнозована характеристика. Отже, можна використати ці знання та сформувати серверні групи або ж об'ємні сервери та пропускні канали для розміщення декількох мікропроцесорів, які в сумі будуть давати навантаження близьке до повного.

Метою цієї статті є показати можливість більш ефективно використовувати енергетичні та обчислювальні ресурси при використанні хмарних технологій шляхом застосування запропонованого в статті алгоритму пошуку доповнюючих навантажень на серверні групи, так щоб завантаженість всіх елементів системи була максимальною, і в той же час збалансованою.

## Характеристики для балансування

Можливість ефективно збалансувати навантаження на серверні групи базується на статистичних даних про використання кожного з видів ресурсу. Нижче наводиться список характеристик, які можуть бути взятими до уваги.

## Мережевий ресурс

Навантаження мережі - кількість трафіку на мережі впливає на стійкість та надійність зв'язку між мікросервісами. Завеликий обсяг трафіку може призвести до перевантаження мережевих ресурсів та зниження швидкості передачі даних. Водночас низький рівень навантаження мережі не виправдовуватиме її ресурс.

Основний показник навантаження мережі - це її пропускна здатність (bandwidth). Вона характеризується кількістю даних, яку можна передати через канал за одиницю часу. Зазвичай вимірюється в бітах на секунду (bps) або його кратному значенні, такому як кілобіт на секунду (kbps) або мегабіт на секунду (Mbps). Чим вища пропускна здатність, тим більше даних може бути передано за одиницю часу.

У випадку проблем з передачею даних, використовується поняття затримки (latency). Це час, необхідний для передачі сигналу від джерела до приймача. Вимірюється в мілісекундах (ms) або мікросекундах (μs). Малий час затримки означає швидку передачу сигналу. Якщо мова йде про одну мережу, затримки під час передачі ма-

лих повідомлень будуть вимірюватись у мікросекундах. З погляду спостерігача, все відбуватиметься блискавично. Більші затримки в тій самій мережі будуть очікуватись у випадку передачі, наприклад, об'ємних файлів між мікросервісами.

Щодо допустимих затримок, існують виміри та розрахунки, в яких межах вони мають бути, щоб користувач їх не відчув. За вимірами компанії Гугл, це затримки до 1.5 секунди, якщо це звичайний вебзастосунок, та 0.5 секунд, якщо відбувається передача відео [3].

Іншою особливістю передачі даних є джиттер (jitter), тобто непостійність в затримці передачі даних через канал. Він вимірюється в мілісекундах (ms) або мікросекундах (μs). Джиттер може вказувати на неналежне керування трафіком або нестабільність мережі. Він може призводити до моментальних пікових навантажень, в той час, як середнє навантаження буде невисоким. Треба слідкувати, щоб джиттери системи були зведені до мінімуму, адже це говоритиме про якість та надійність системи в цілому.

Незалежно від особливостей передачі даних, дуже важливою характеристикою є втрата пакетів (packet loss), з якою можна боротись за допомогою fault tolerance методів [4]. Вона вимірюється у відсотках пакетів, які не були доставлені до призначеного місця через помилки в мережі або перевантаження. Велика втрата пакетів може призвести до погіршення якості передачі даних. Через можливу втрату пакетів доводиться передбачувати механізм повторного посилання інформації [5], якщо вона ще лишається актуальною. У добре розробленій системі втрати пакетів не повинно бути, а повторне відправлення має відбуватись тільки в разі фізичного розриву мережі. Допустимим вважається показник менший за 1% втрачених пакетів [6]. При передачі, наприклад, аудіо чи відео не буде суттєво відчутно проблем зі зв'язком в такому випадку.

Всі вищезгадані характеристики беруться до уваги за умови повної доступності (availability) каналу, тобто готовності каналу передавати дані [7]. Ця характеристика пропорційна та розподілятиметься між декількома мікропроцесорами в необхідній долі залежно від їх середніх очікуваних показників навантаження на мережу.

## Процесорна потужність

Завантаження процесорної потужності - це процес, під час якого процесор обробляє всі запити, що надходять. Ефективність цього процесу залежить від стилю написання коду, використання реактивного програмування або віртуальних потоків. Незалежно від цього, кожен запит приймається та опрацьовується відповідним потоком.

Для оцінки потреби в процесорній потужності для даного застосунку варто враховувати показники завантаженості ядра процесора або сумарний середній показник завантаженості групи ядер за одиницю часу.

Розглядаючи вхідні параметри, важливо враховувати не лише середню очікувану завантаженість процесора протягом певного часу, а також мінімальну, бажану та в деяких випадках максимальну кількість ядер-потоків сервера. Ці параметри важливі, оскільки багатопотокові застосунки можуть бути оптимізовані для певної кількості потоків. Розбіжність кількості потоків з вимогами розробленого коду може призвести до зниження ефективності багатопотокової розробки або навіть до виникнення негативних наслідків. Наприклад, якщо комп'ютеру доведеться обробляти дві задачі одночасно, одним потоком, ефективність може значно зменшитися через постійне перемикання між ними.

## Пам'ять

Мають значення як оперативна, так і постійна (дискова). Оперативної пам'яті має вистачати для потреб обробки запитів, а жорсткий диск має виконувати скоріше функцію запасної оперативної пам'яті, куди відбуватиметься тимчасове вивантажування інформації з оперативної в разі пікового навантаження. Основна функція постійної пам'яті - слугувати операційній системі для розміщення необхідних файлів та хостити мікросервіси. Самі по собі мікросервіси не мають записувати на дискову

пам'ять свої дані, адже вся необхідна інформація радше має зберігатись у під'єднаних базах даних.

Приклад вхідних даних

У Табл.1 міститься приклад характеристик, зібраних з робочої програми на одному мікропроцесорі за одну годину роботи, додатково із загальними характеристиками мікропроцесора, які надаються розробниками ПЗ. Погодинні характеристики вказані своїм середнім значенням. Такі характеристики мають надходити з кожним мікропроцесором за кожну годину, тобто 24 значення кожної змінної характеристики. Якщо певні дні тижня мають суттєво інші показники, їх теж слід надати.

Таблиця 1

Зведені характеристики загальні до мікропроцесора та середнє значення за конкретну годину

| Назва характеристики | Особливості | Значення |
|---|---|---|
| Мережеве навантаження | Пропускна здатність - 10 Мбіт/с (ethernet) | 8 Мбіт/с |
| Процесор | 2,5 GHz Quad-Core Intel Core i7 | 68% |
| Ядра | Quad-Core | 4 |
| Оперативна пам'ять (RAM) | 8 GB 1600 MHz DDR3 | 4 Гб |
| Жорсткий диск мінімальний (HD) | Seagate BarraCuda HDD | 60 Гб |

## Алгоритм роботи системи пошуку доповнюючих екземплярів мікросервісів

Алгоритм пошуку доповнень має бути гнучким, адже ідеального доповнення може не існувати, тому основні керівні моменти - це обрана допустима похибка неповноти. Задача повного перебору була б допустима, якби розраховувати на сотні екземплярів мікросервісів, при тисячах слід вводити оптимізації.

Алгоритм пошуку доповнення має запускатись у разі надходження нових мікросервісів для розподілення, включаючи всі наявні мікросервіси, які неоптимально запущені. Також зі зміною вхідних табличних показників, має сенс теж переглянути доцільність групування певних екземплярів. Водночас цей випадок зводиться до перегляду неоптимально налаштованих серверних груп, що можна буде спостерігати по статистиці кожної групи.

Під статистикою по мікропроцесору слід розуміти загальне використання ресурсів усіх запущених екземплярів. Тобто, якщо екземплярів 5, і кожен використовує 4 ядра, то сумарно ядер буде 20. Натомість статистика по серверній групі враховує сумарну характеристику конкретних запущених екземплярів процесорів і конкретну статистику кожного екземпляра.

## Підготовка даних

Проранжувати мікропроцесори за основною та додатковими характеристиками від найбільшого до найменшого. Під час вибору мікропроцесорів для суміщення, початково треба дивитись на ті, в яких значення максимального навантаження приблизно рівні, водночас графік завантаження зміщений на фазу. Іншими словами, якщо один завантажений на 500 Флопс вдень і на 100 вночі, а інший - навпаки.

Сортування знадобиться для підбору схожих навантажень мікропроцесорів. Також воно дасть можливість зручніше та швидше, не виконуючи додаткового перебору, зробити початкові корекції з мікропроцесорами перед запуском алгоритму.

Задача підготовки даних полягає у **розподілі великих мікропроцесорів.** Слід розподілити мікропроцесори із занадто великим завантаженням та апріорі поділити їх на 2 або більше частин, щоб обмежити максимально допустиме використання ресурсів. Вибір кількості частин слід здійснювати, базуючись на максимально допустимому значенні за заданою характеристикою $\underline{P_{boundary}}$. Якщо наявні частини, які сильно виділяються завантаженістю поміж іншими, їх слід відокремити в нові екземп-

ляри мікропроцесорів, які будуть запускатися тільки в заданий час.

Зручно мати певну кількість мікропроцесорів, котрі завжди працюватимуть на максимум, і в разі необхідності запускати помічників. Це краще, ніж декілька постійних мікропроцесорів, що працюють не на повну потужність.

Таким чином, після поділу містких мікропроцесорів їхній графік завантаження стане зовсім іншим. У одних він буде однаковий протягом усього дня, а у інших буде частковий, тобто завантаженість у певні години та нулі в інші. Це дозволить не шукати пару повноцінним мікропроцесорам та гнучко підбирати компаньйонів частково завантаженим. Ідея доповнюючих мікросервісів детально розписана у статті [8].

Нижче наведено математичний вираз (1), який описує необхідність поділу максимального навантаження $\underline{P_{\max}}$ на кількість частин так, щоб кожна поділена частина була дещо меншою за $\underline{P_{\text{boundary}}}$:

$$\underline{\frac{P_{\max}}{m} < P_{\text{boundary}}}, \qquad (1)$$

де $\underline{P_{\max}}$ - максимальне навантаження по заданій характеристиці, $\underline{P_{\text{boundary}}}$ - граничне значення навантаження на кожну частину, $\underline{m}$ - ціле додатне число, кількість частин, на які потрібно поділити максимальне навантаження. Дана формула та наступні наведені є авторськими та випливають з постановки задачі та способу її вирішення.

## Пошук закономірностей

Якщо у добовому використанні процесора спостерігаються виражені максимуми та мінімуми, можна вивести закономірності його роботи. Знання значень періоду, амплітуди та фази допоможуть у пошуку антипода. Таким чином, до табличних характеристик слід додати помітки для виділення

- мінімумів,
- максимумів завантаженості,
- значення завантаженості.

Такі помітки можна зробити у рядках навпроти відповідного часу.

Також ці характеристики слід занести до окремої таблиці для розрахунку доповнень, вказавши ідентифікатор екземпляру мікропроцесора. Отримавши дані про всі екземпляри, можна згрупувати мікропроцесори за схожою поведінкою. Це полегшить пошук доповнень.

## Ранжування

У межах групи проранжувати особини за максимумами використання кожної характеристики, виділяючи основну. Це знову ж таки пришвидшить підбір пари, позбавляючи необхідності повного перебору в межах групи, адже так можна ефективніше підбирати екземпляри, щоб доповнити навантаження, якого не вистачає.

## Нормалізація навантаження

Почати пошук найкращого доповнення з сумісної групи за наявності. Там, де приблизний максимум одного екземпляра, має бути приблизний мінімум або не максимум іншого.

У процесі суміщення першим етапом є нормалізація спільного максимального завантаження за ключовим або екстремальним параметром. Таким параметром є найбільша характеристика з таблиці, тобто або процесорне використання, або мережевий ресурс, або оперативна пам'ять.

Під нормалізацією мається на увазі пошук сумарної найбільшої завантаженості в кожний момент часу. Це не обов'язково буде накладення максимального навантаження, оскільки в перетині можуть траплятись мінімальні, тому треба перевірити всі значення в кожен вимір. Нормалізована величина, або ж сумарне максимальне завантаження, буде взята як повна сумарна завантаженість, тобто 1 або 100%. Це значення буде потужністю серверної групи за обраним параметром.

Формула для знаходження часового інтервалу $t_{max}$, за якого сума навантажень $P_i$ за заданим параметром буде максимальною, може бути записана наступним чином (2):

$$\underline{t_{\max} = \underset{t}{\operatorname{argmax}} \sum_{i=1}^{n} P_i}, \qquad (2)$$

де $t_{\max}$ - шуканий часовий інтервал, який максимізує суму навантажень, $\underset{t}{\operatorname{argmax}}$ - оператор, який знаходить значення параметра $t$, за якого вираз $\sum_{i=1}^{n} P_i$ досягає максимуму, $P_i$ - навантаження на і-тому мікропроцесорі, $n$ - ціле додатне число $n \in \mathbb{Z}^+$, яке представляє кількість мікропроцесорів, які розглядається для однієї серверної групи.

Таким чином, формулу результату нормалізації можна записати у вигляді (3).

$$P_{t_{max}} = \sum_{i=1}^{n} P_{i,t_{max}} = 1 = 100\% \quad , \tag{3}$$

## Розрахунок недовикористаного ресурсу

Отримавши нормалізовані дані, можна виміряти кількість недовикористаного ресурсу та залежно від показників ухвалити рішення про доцільність такої комбінації мікросервісів. Недовикористаний ресурс $U$ являє собою суму різниць нормалізованого максимального значення $P_{t_{max}}$ з сумою $P_i$ за всі години. Це виражається формулою (4):

$$U = \sum_{t=1}^{T} (P_{t_{max}} - \sum_{i=1}^{n} P_i) \quad , \tag{4}$$

де $T$ - кількість годин, $P_{t_{max}}$ - нормалізоване максимальне значення ресурсу, використаного всіма екземплярами потенційної серверної групи, сума $P_i$ - значення спільно використаного ресурсу $P$ усіма екземплярами $i$.

## Умови продовження та завершення пошуку доповнення

Зважаючи на значення $U$, рішення можуть бути наступні:

а) **Доповнення високого рівня**, тобто майже всі ресурси використовуються на повну в будь-який час, і групу сформовано. Критерій повноцінного використання можна визначити через формулу (5), представлену нижче, яка являє середнє значення суми недовикористаного ресурсу $U_t$ за весь час T, яке має бути меншим або рівним $U_{\text{low}}$.

$$\frac{1}{T} \sum_{t=0}^{T} U_t \leq U_{\text{low}} \quad , \tag{5}$$

де $T$ - кількість годин, $U_t$ - дискретне значення недовикористаного ресурсу в час $t$, $U_{\text{low}}$ - мінімально припустиме значення недовикористаного (unused) ресурсу.

б) **Доповнення неефективне**. В цьому варіанті повне процесорне використання зустрічається в рідкісні часові інтервали, але і можливостей вмістити додаткові мікросервіси теж мало, тому краще шукати інше доповнення. $U_{high}$ - рівень невикористаного (unused) ресурсу, який вважається завеликим. Вираз (6), який це описує, наведено нижче.

$$U_{\text{low}} < \frac{1}{T} \sum_{t=0}^{T} U_t \leq U_{\text{high}} \tag{6}$$

в) **Доповнення незакінчене** (7), тобто є можливість доповнення між заданими мікросервісами тільки, якщо додаткові мікросервіси будуть додані, адже недовикористаний ресурс занадто великий. Тоді слід знайти мікросервіс з малою потребою в ресурсах і додати його до групи, повторюючи всі кроки заново.

$$\frac{1}{T} \sum_{t=0}^{T} U_t > U_{\text{high}} \tag{7}$$

Найкращим варіантом вбачається перший, коли одразу чи після додавання декількох мікросервісів знайдено повне доповнення. Водночас слід розуміти, що така ситуація може статися не з усіма мікропроцесорами. Після виконання деякої кількості переборів A (attempts) з певним мікропроцесором, потрібно вибрати найменш неуспішну комбінацію, або ж погодитись, що цей мікропроцесор буде самостійно на сервері. Це теж слід оцінити порівнюючи кількість недовикористаного ресурсу U. Для цього програма розподілу буде запам'ятовувати це значення на кожній ітерації, занотовуючи набір суміщених екземплярів та значення невикористаного ресурсу, поки не буде ухвалено рішення.

## Приклад з візуалізацією

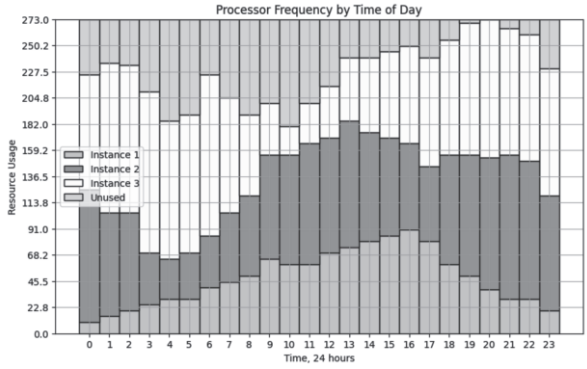


Рис. 1. Приклад суміщення 3х екземплярів мікропроцесорів в одну групу з відміченим недовикористаним ресурсом.

На Рис.1 показано 2 ітерації пошуку доповнюючих екземплярів мікропроцесорів. На першій ітерації зійшлись екземпляр 1 - позначений сірим, та екземпляр 2 - позначений червоним. У місці максимуму першого, у другого був локальний мінімум, тому вони могли бути скомбіновані, як зазначено у пункті "Пошук закономірностей". Водночас, спостерігається очевидний сильний мінімум та завеликий діапазон невикористаності ресурсу, описаний у формулі (7). Тому алгоритм продовжив далі пошук доповнення.

Маючи максимум в зоні мінімума сумарної функції двох перших екземплярів, була спроба під'єднати третій екземпляр іншого мікросервісу, позначеного жовтим. Після спроби система увійшла в стан (6). Дане значення буде внесено в реєстр та пошук продовжено з надією знайти кращу допасованість, яка б мінімізувала значення *U*.

## Висновки

У висновках статті пропонується алгоритм, який дозволить оптимізувати використання ресурсів хмарних систем шляхом ефективного розподілу навантаження між різними мікросервісами. Цей алгоритм може бути корисним для постачальників хмарних послуг та організацій, які використовують хмарні середовища для розгортання своїх додатків.

Запропонований підхід дозволяє максимізувати використання серверних ресурсів, зменшуючи час простою серверів та оптимізуючи роботу мікросервісів. Це своєю чергою сприятиме покращенню продуктивності та зниженню витрат на обслуговування хмарних інфраструктур.

Враховуючи те, що використання хмарних ресурсів стає все більш поширеним і важливим для багатьох організацій, розробка ефективних методів оптимізації цих ресурсів має велике значення для підвищення конкурентоспроможності та забезпечення стабільності бізнесу.

## Література

1. "A review of in-memory computing for machine learning: architectures, options," *Int. J. Web Inf. Syst.*, Dec. 2023, doi: 10.1108/IJWIS-08-2023-0131.
2. Z. Li, D. Seco, and A. Sánchez Rodríguez, "Microservice-Oriented Platform for Internet of Big Data Analytics: A Proof of Concept," *Sensors*, vol. 19, no. 5, p. 1134, Mar. 2019, doi: 10.3390/s19051134.
3. "What is Latency: The Hitchhiker's Guide," Obkio. Accessed: Mar. 26, 2024. [Online]. Available: https://obkio.com/blog/what-is-latency/
4. O. Dmytrenko and M. Skulysh, "Fault Tolerance Redundancy Methods for IoT Devices," *Infocommunication Comput. Technol.*, vol. 2(04), no. University "Ukraine," pp. 59–65, Dec. 2022.
5. O. Dmytrenko and M. Skulysh, "Handling with a Microservice Failure and Adopting Retries," presented at the Pan-Ukrainian science-practical conference of students, postgraduates and young scientists "Theoretical and Applied Problems of Physics, Mathematics and Informatics," Kyiv, Ukraine: ФІЗТЕХ, Jun. 2022, pp. 194–196. [Online]. Available: https://drive.google.com/file/d/1MbM9YqnSitNNdCp0CZ1yguDHweWth3TK/view
6. "What Is Packet Loss: The Invisible Enemy of Networks," Obkio. Accessed: Mar. 26, 2024. [Online]. Available: https://obkio.com/blog/what-is-packet-loss/
7. H. Zhang, W. Yang, H. Ji, X. Li, V. C. M. Leung, and L. Yang, "Delay Aware

Resource Allocation for Device-to-Device Communication Underlaying Cellular Networks," in *Wireless Internet*, M. Huang, Y. Zhang, W. Jing, and A. Mehmood, Eds., Cham: Springer International Publishing, 2018, pp. 207–217. doi: 10.1007/978-3-319-72998-5_22.

8. O. Dmytrenko and M. Skulysh, "Method of Grouping Complementary Microservices Using Fuzzy Lattice Theory," vol. 12, no. 1, pp. 11–18, Mar. 2024, doi: 10.25673/115636.

Одержано: 17.04.2024

Внутрішня рецензія отримана: 25.04.2024

Зовнішня рецензія отримана: 26.04.2024

***Про авторів***:

[1]*Дмитренко Олександра Анатоліївна* ,
асистент, аспірант.
https://orcid.org/0009-0009-4785-422.

[1]*Скулиш Марія Анатоліївна*
Кандидат технічних наук,
професор
http://orcid.org/0000-0002-5141-1382.

***Місце роботи авторів:***

[1]Національний технічний університет України "Київський політехнічний інститут ім. Ігоря Сікорського"
тел. +380 44 204-94-94
E-mail: webmaster@kpi.ua
Сайт: https://kpi.ua/