УДК 004.75
DOI: https://doi.org/10.17721/AIT.2024.1.08

Олександра ДМИТРЕНКО, асп., асист.
ORCID ID: 0009-0009-4785-4226
e-mail: o_dmytrenko@lll.kpi.ua
Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна

Марія СКУЛИШ, д-р техн. наук, проф.
ORCID ID: 0000-0002-5141-1382
e-mail: mskulysh@gmail.com
Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна

ГРУПУВАННЯ МІКРОПОСЛУГ ДЛЯ ВИКОРИСТАННЯ НЕПОВНО ЗАЛУЧЕНИХ РЕСУРСІВ

В с т у п . ***Завдання оптимізації використання ресурсів хмарних систем має високий пріоритет, оскільки ця технологія стала широко розповсюдженою та легкодоступною. Поширеною практикою є створення ефективних сервісів, які легко адаптуються до різних типів продуктів, наприклад, безсерверні (serverless) технології, пропрієтарні бази даних і програмне забезпечення, оптимізоване для хмари. Тестування продуктивності хмарних застосунків стало значно простішим із появою Docker і Kubernetes, що додатково збільшило попит на хмарні ресурси. Оскільки значна кількість ресурсів витрачається на розроблення й обслуговування хмарної інфраструктури, важливо розуміти, як ці ресурси можна оптимізувати.***

М е т о д и . ***Основна ідея полягає в реорганізації хмарної інфраструктури так, щоб програмне забезпечення розподілялося по серверних групах, а не серверах, за принципом комплементарності, що означає наближення навантаження на апаратне забезпечення до повного. Для формування комплементарних груп використовують методи кластеризації (K-Means), задача про множинний рюкзак, жадібний алгоритм, сортування, бінарний пошук, перемішування та поділ на основі середнього значення та стандартного відхилення. Додатково для візуалізації пропонують алгоритми FastDTW та Z-масштабування.***

Р е з у л ь т а т и . ***Розглянуто ключові характеристики комп'ютерних систем, які можна виміряти та на які можна впливати. Ці характеристики включають пропускну здатність каналів передачі, оцінку продуктивності на основі затримки, обсяг оперативної та постійної пам'яті, обчислювальну потужність і кількість ядер. Запропоновано алгоритм для визначення комплементарних екземплярів мікросервісів, які могли б ефективно використовувати серверні ресурси. Спочатку екземпляри мікросервісів класифікують за їх місткістю ключового ресурсу.***

В и с н о в к и . ***Запропоновано алгоритм, який оптимізує використання ресурсів хмарної системи за допомогою ефективного розподілу навантаження доповнених мікросервісів між різними серверними групами. Оптимізація полягає в максимізації застосування серверних ресурсів шляхом заповнення його вільного часу іншими мікросервісами й, отже, зменшенням простою серверів. Це, своєю чергою, сприятиме покращенню загальної продуктивності хмари та зменшенню витрат на обслуговування хмарних інфраструктур. Наведений підхід застосовують як до мікросервісів, так і до монолітів, з описаними мінімальними змінами. Цей алгоритм може бути корисним для постачальників хмарних послуг та організацій, які використовують хмарні середовища для розгортання своїх застосунків.***

К л ю ч о в і с л о в а : ***групування мікросервісів, процесорна ефективність, оптимізація використання енергії, хмарні системи, мікросервісна архітектура.***

Вступ

Еволюція хмарних технологій започаткувала новий виток у розробленні програмного забезпечення на початку 2000-х рр. Справжній прорив у розробленні та тестуванні хмарних застосунків стався 2013 р. із появою Docker. Упровадження технології контейнеризації, а згодом і системи оркестрації Kubernetes, створило сприятливе середовище для розвитку мікросервісної архітектури, надавши можливість ефективно моделювати роботу та масштабування мікросервісів у хмарному середовищі.

Мікросервісна архітектура демонструє особливу ефективність у масштабних застосунках (Aspire Systems, 2023), де різні функціональні компоненти можуть мати різний ступінь взаємозалежності. Характерною особливістю користувацької поведінки є нерівномірність використання різних функцій застосунку – деякі компоненти можуть потребувати значно більше ресурсів, ніж інші. Мікросервісний підхід дозволяє здійснювати вибіркове масштабування, запускаючи додаткові екземпляри лише тих компонентів, які зазнають підвищеного навантаження. Водночас і монолітні застосунки можуть запускатись у кількох екземплярах, що сприяє перерозподілу навантаження та полегшує питання обслуговування системи. Такий підхід забезпечує економічну ефективність, оскільки точкове масштабування окремих компонентів потребує менше ресурсів порівняно з розгортанням додаткових копій усього застосунку у використанні балансування навантаження (Sharma, 2023).

Для забезпечення надійності хмарної інфраструктури, оновлення серверного обладнання зазвичай планують відповідно до закінчення гарантійного терміну. Це створює передумови для максимально ефективного використання наявних обчислювальних ресурсів протягом усього життєвого циклу обладнання, адже інакше воно не окупиться.

У цій статті запропоновано алгоритм, спрямований на оптимізацію розподілу навантаження між мікросервісами й ефективне використання серверних і мережних ресурсів хмарної інфраструктури.

Типовим для серверних застосунків є нерівномірне навантаження, що залежить від специфіки застосунку та часових патернів використання. Передбачуваність цих патернів створює можливість для оптимізації – формування серверних груп або кластерів, де комбінація різних мікросервісів забезпечує близьке до оптимального сумарне навантаження. Як результат – підвищення енергетичної та обчислювальної ефективності хмарних систем впровадженням алгоритму пошуку комплементарних навантажень для роботи на серверних групах.

Постановка задачі. Розробити алгоритм для завантаження максимальної кількості серверних ресурсів хостингової системи (хмарного провайдера) через оптимізацію кількості виконуваних задач шляхом пошуку доповнень до мікросервісів і монолітів (як підмножини мікросервісів), що необхідно розмістити на серверах.

Мета дослідження. Продемонструвати можливість повного ефективного завантаження серверів, що сприятиме зменшенню їхньої кількості. Спрямувати підхід на досягнення максимальної та збалансованої завантаженості всіх компонентів системи.

Методи

В цьому розділі представлено алгоритм для пошуку комплементарних або доповнювальних навантажень, щоб кілька елементів програмного забезпечення (ПЗ) могли сформувати повне навантаження на серверну групу, таким способом використовуючи максимально повно її ресурс. На створення алгоритму надихнула нечітка теорія ґраток, зокрема поняття доповняльних ґраток (Dmytrenko, & Skulysh, 2024a) та лінійне програмування. Елементи ПЗ, мікросервіси чи моноліти, для спрощення будуть надалі називатись мікросервісами, адже і до монолітів можна ставитись як до мікросервісів за використання скейлінгу, що охоплює запуск додаткових екземплярів з балансуванням навантаження між ними. У випадках, коли поведінка мікросервісів і монолітів відрізнятиметься, слідуватиме додаткове пояснення.

Алгоритм наведено по пунктах, у деяких пунктах є посилання на інші, тому номери допоможуть зорієнтуватись.

1. **Аналіз вхідних мікросервісів.** Вхідні дані: статистика стосовно роботи кожного мікросервісу або моноліта, який сприйматиметься як мікросервіс. Статистика містить такі дані:
   - CPU (очікуване завантаження в інтервал часу, кількість ядер та їхня частота);
   - RAM (очікуване завантаження та загальна необхідна кількість);
   - канальний ресурс – channel (інтервальний очікуваний об'єм трафіка і загальна пропускна здатність каналу);
   - базові характеристики техніки, на якій виконувались випробування, такі як: кількість ядер, вимоги до жорсткого диска;
   - прапорець можливості поділу чи реплікації вхідного елемента ПЗ canBeChunked;
   - географічна зона для роботи мікросервісу. Вона може задаватись на етапі вхідних даних замовником, або ж вираховуватись у процесі роботи ПЗ на клауд-системі й оновлюватись на наступних ітераціях алгоритму.

2. **Нормалізація вхідних даних.** Перед початком роботи з ПЗ, яке прийшло з іншої системи, варто перевести метрики з іншого середовища на підходящі серверні потужності клауд-провайдера.

3. **Поділ великозатратних елементів ПЗ.** Поділ повного необхідного ресурсу на менші можливий для мікропроцесорів, а також переважно можливий і для монолітів. Для цього в початковій умові може задаватись параметр "canChunk", тобто можливість поділу. На практиці "поділом" називають scaling – масштабування, тобто створення додаткових екземплярів мікросервісів за надмірного навантаження або ж дублювання монолітів. Для розв'язання задачі ефективного використання ресурсів, зручніше оперувати поняттям повного та часткового навантаження, тому замість "розширення" використовують слово "поділ".

Поділ здійснюють на основі найбільшого, тобто ключового показника (CPU, RAM чи канальний ресурс). Жоден із показників не має перевищувати дозволеної норми. Після поділу ключовим показником стає новий найбільший показник. Навантаження мікросервісу в час його найбільшого піка має бути меншим за граничне значення

$$P_{t_{\max}} \le P_{boundary} - P_{boundary} \times \alpha_{boundary},$$

де $\alpha_{boundary}$ – прийнятний коефіцієнт недоповненості навантаження, яке може бути рівним $D_{c_{\max}}$ (див. пункт 9), тобто нехай понад 70–80 %, то до нього буде важко знайти доповнювальний. Такий мікросервіс потрібно теж помітити як габаритний.

4. **Систематизація всіх мікросерверів, що потребують серверного розміщення.** Поділ може відбуватись на основі активності роботи в межах часових проміжків. Створення додаткового екземпляра відбуватиметься тільки на певний період часу, коли буде перевищено максимальне допустиме значення за якоюсь характеристикою з пункту 1:

$$P_{t_{\max}} + \alpha_{reserve} \cdot P_{t_{\max}} \ge P_{boundary}.$$

Поділ відбуватиметься на стільки частин m, скільки повних максимальних завантажень з урахуванням безпечного резерву $\alpha_{reserve}$ може вміститись у заданий елемент, плюс те навантаження, яке не буде повним, але залишковим від ділення, округлений до більшого:

$$m = \left\lceil \frac{P_{\max} + m \cdot \alpha_{reserve}}{P_{boundary}} \right\rceil.$$

У випадках, коли той самий мікросервіс необхідно запустити на кількох географічних зонах, він не буде сприйматись як той самий мікросервіс із погляду алгоритму. Якщо однаковий мікросервіс деплоїться (інсталюється) на кілька географічних зон, щоб швидше забезпечувати відповідь клієнтам, розташованим у цих зонах, попередньо необхідно зробити заміри використання потужностей, що будуть відповідати цим зонам. Такі заміри можуть бути наведені у вхідних даних або обчислені на наступних ітераціях клауд-провайдером.

На виході: поділені мікросервіси з новими ідентифікаторами, розподілено за географічними зонами. Кожна географічна зона містить такий список груп:

1) список мікросервісів вільного розподілу, тобто мікросервіси, які можна комбінувати з будь-якими іншими;

2) список списків мікросервісних груп, які мають розміщуватись разом на підставі використання спільного ресурсу або питань безпеки;

3) список мікросервісів, які вимагають окремого серверного ресурсу, адже їхні потреби завеликі, навантаження непередбачуване, або через питання безпеки.

5. **Нормалізація шаблона роботи мікросервісу.** Нормалізація виконується через Z-Score алгоритм (Ozdemir, & Susarla, 2018) з метою пошуку класів еквівалентності. Ідея полягає в записі статистики використання мікросервісом кожного ресурсу у проміжку, який приблизно рівний [–2; 2]. Середні значення показників розташуються близько нуля, менші за середні – у від'ємній площині, а більші за середні – у додатній. Якщо показник дуже відрізняється від інших, може бути сенс поділити цей мікросервіс на 2 чи більше, щоб створювати виділений екземпляр на екстремальний час

ISSN 2788-6603

для оброблення підвищеної кількості запитів. В разі проблем із підбору доповнювального мікросервісу, це буде зроблено на пізньому етапі алгоритму (Dmytrenko, Skulysh, Globa, 2024).

6. **Візуалізація шаблонів роботи.** Як опційний крок можна розглядати візуалізацію результату порівняння схожості зразків роботи мікросервісів, прораховану за допомогою FastDTW (Salvador, & Chan, 2004) алгоритму, який накладає часові ряди в оптимальному місці (Dmytrenko, Skulysh, Globa, 2024).

7. **Пошук подібних шаблонів роботи за ключовим критерієм.** Наступним кроком є пошук схожих шаблонів роботи ПЗ. Основною задачею пошуку схожості шаблонів є виділення класів, подібних за стилем роботи мікросервісів, користуючись ключовим (найвитратнішим) критерієм. Переважно це CPU, а канальний ресурс і RAM перевіряють додатково як ті, що не накладатимуться. Якщо ж для частини мікросервісів ключовий критерій інший, то їх варто обробляти окремо за тим самим принципом.

Для пошуку схожості шаблонів використовують алгоритм кластеризації KMeans. За цим алгоритмом формують центроїди – точки, які представляють центри груп мікросервісів схожої поведінки. Оскільки статистика використання мікросервісу включає значення за певною характеристикою за цілий день, то і точка-центроїд теж являє собою часовий ряд значень обраної характеристики.

8. **Двовимірна візуалізація кластерів часових рядів.** З метою візуалізації можна представити часовий ряд як двовимірну точку користуючись методом головного компонента – Principal component analysis (PCA). Таке представлення дасть можливість побачити групи дочок та їхні центроїди та зробити висновки про якість зон скупченості. На рис. 1 ліворуч за допомогою PCA зображено поділ мікросервісів, зазначених у пункті 11, на кластери, де центроїд позначено хрестиком, а мікросервіси – точками. Праворуч показано ті самі часові ряди у звичайному вигляді. Пунктирною лінією позначені центроїди. Детальнішу інформацію можна прочитати в роботі (Dmytrenko, Skulysh, Globa, 2024).

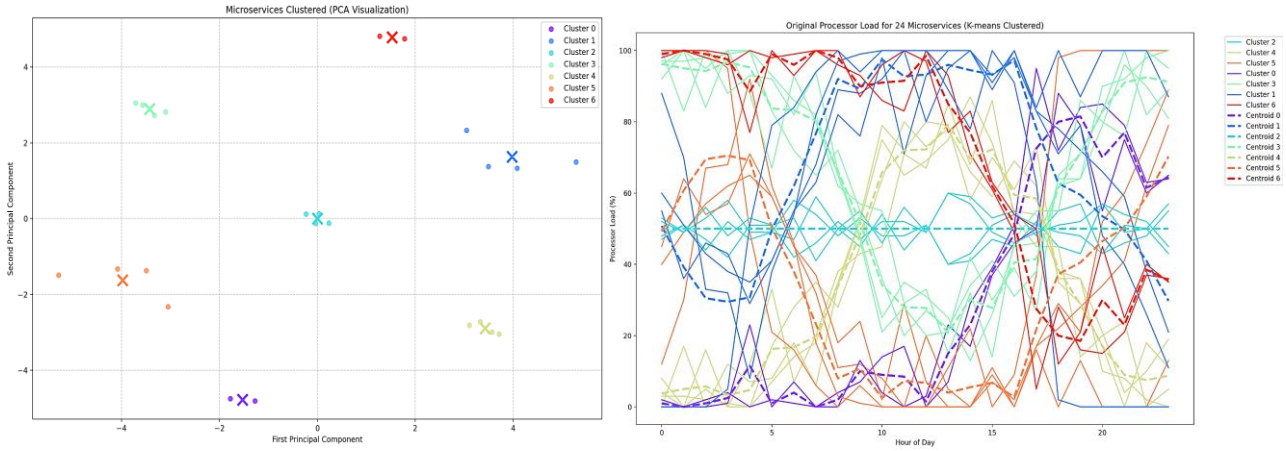

Рис. 1. Ліворуч: поділ мікросервісів на кластери методом K-Means і зображення у двовимірному вигляді за допомогою PCA методу. Праворуч: поділ тих самих кластеризованих мікросервісів методом K-Means у вигляді часових рядів

9. **Пошук доповняльних груп.** Кульмінаційним моментом алгоритму є пошук доповняльних груп мікросервісів і подальший пошук доповнювальних мікросервісів (пункт 10).

За пошуку доповнювальних груп вираховують показник розбіжності центроїдів $D_{ci,j}$ за ключовим параметром, як нормовану різницю між максимальним завантаженням релевантної серверної групи $sum_{\max_{i,j}}$ та сумою навантажень, яке зроблять обидва центроїди.

$$D_{ci,j} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{sum_{\max_{i,j}} - \left(vector_i + vector_j\right)}{sum_{\max_{i,j}}}\right).$$

Якщо результуюче значення роозміщене в межах заданої похибки $D_{ci,j} \le D_{\max_{i,j}}$, то групи вважають доповняльними. Результатом цього кроку є список проранжованих доповнень до кожного кластера.

10. **Пошук доповнювальних пар мікросервісів.** При здійсненні пошуку доповнених мікросервісів використовуються початкові метрики кожного екземпляру, а не нормовані. З двох найбільш доповнювальних груп береться по одному мікросервісу *i* та *j* з найближчою різницею в амплітуді. Щоб не перевіряти заздалегідь неробочі варіанти, можна порівняти різницю хвилі потужності, вирахувану як різниця потужностей у час максимуму $t_{\max}$ та час мінімуму $t_{\min}$, та переконатись, що вона знаходиться в певній умовою заданій межі $\Delta_{similarity}$:

$$\left(P_{i,t_{\max}} - P_{i,t_{\min}}\right) - \left(P_{j,t_{\max}} - P_{j,t_{\min}}\right) \le \Delta_{similarity}.$$

Мікросервіси накладають, як на рис. 2, та оцінюють сумарну невідповідність (Discrepancy) $D_{i,j}$ за кожним показником, тобто кількості недовикористаних ресурсів. Якщо цей показник менший за граничний, заданий у вхідній умові, за ключовим показником (напр., D_{CPU} – невідповідність, дозволена для CPU), а також нема перетинів за іншими показниками (канальний ресурс і RAM), то вибрані мікросервіси формують групу. Вони видаляються з груп пошуку доповнювальних мікросервісів, адже стають закритими для пошуку іншої пари.

Під їхню комбінацію буде виділено серверну групу, загальний ресурс якої P_{group_g} рівний їхньому максимальному ресурсу $P_{\max_g}$ з урахуванням заданої початковою умовою похибки $\alpha_{reserve}$ (alpha) – відсоток запасу ресурсу. Числове значення обчислюють як відсоток від сумарного найбільшого завантаження в одиницю часу t. У наступній формулі i та j – індекси мікропроцесорів, які входять у групу. Ця формула валідна для довільної кількості елементів у групі g.

$$P_{\max_g} = \max\left(\sum_{g \in i,j} P_g(t)\right).$$

$$P_{group_g} = P_{\max_g} + \alpha_{reserve} \cdot P_{\max_g}.$$

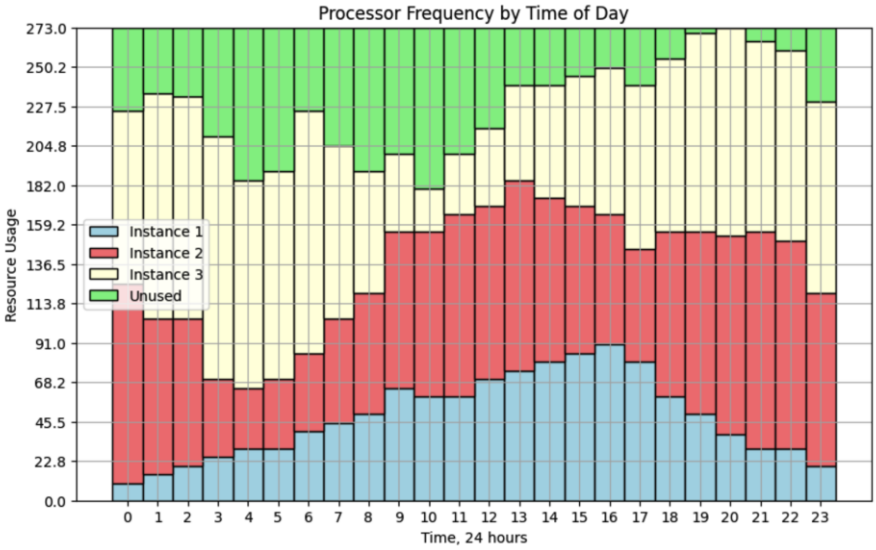

Рис. 2. Навантаження трьох мікросервісів (знизу). Порожній ресурс (зверху)

Якщо ж невідповідність $D_{i,j}$ більша за максимально дозволену $D_{g\ \max}$, обидва мікросервіси продовжують шукати собі пару, помічаючи пройдені мікросервіси як непідходящі.

Пошук пропонується здійснювати після сортування початкової групи мікросервісів, а також використовувати бінарний пошук.

Якщо в межах найкращої доповнювальної групи не було знайдено пару, варто взяти наступну за рейтингом підходящу групу та шукати пар групи не було знайдено пару, варто взяти наступну за рейтингом підходящу групу та шукати пару там. Процес повторюється, поки група, для мікросервісів якої шукали пари, стане порожньою, або ж поки не будуть проаналізовані всі доповнювальні групи.

Мікропроцесори, що не знайшли пари на цій ітерації, записують в окремий масив тих, які підуть на наступну ітерацію.

11. **Повторення ітерацій пошуку.** Якщо лишаються мікросервіси, які не знайшли доповнювачів, варто повторити кроки 7–10. Ітерації повторюють, поки утворюються пари. З кожною ітерацією кількість груп у пункті 7, на які діляться мікросервіси, зменшується пропорційно кількості екземплярів. Для вибору кількості кластерів є тільки емпіричні методи, такі як метод ліктя, силуетний аналіз, Gap-статистика та правило $\sqrt{n}$. Шаблони поведінки мікросервісів слід поділити на такі категорії для великої вибірки:

- робота в денний час: 9–17;
- робота в нічний час: 23–6;
- робота у вечірні години: 17–24;
- робота у ранкові години: 6–11;
- стабільне завантаження протягом доби;
- кілька варіацій завантаження власного типу (2–3).

Сформуємо 7–8 кластерів. Така кількість є резонною для великої вибірки. Якщо у групі лишатиметься менше ніж 2–3 елементи, нема сенсу створювати групу для такої малої кількості. Отже, пропонується користуватись правилом $\sqrt{n}$ у вибірці, що менша 8^2, а в більшій вибірці брати завжди 8 або 7 за умови регіонального розподілу мікросервісів.

ISSN 2788-6603

Останнє важливо, оскільки в інших країнах у разі зсуву часу робочі години можуть бути інші, тому аби коректніше знаходити доповнення, можна виділити більше груп кластеризації.

12. **Доповнення неможливі.** Може трапитися, що не всі мікросервіси знайшли доповнювальні навіть після 11-го кроку. Це може статись у таких випадках:

1) характер роботи мікросервісу випадковий. В такому разі йому варто виділити окремий сервер і не намагатись його сумістити з іншими;

2) екстремуми роботи мікросервісу дуже різкі, або ж у нестандартний час. В такому випадку пропонується поділити цей мікросервіс на кілька, якщо виражених екстремумів не більше двох, що описано в наступному пункті. В іншому разі – виділити окремий сервер під цей мікросервіс;

3) мікросервіси замалі, тому сумарне доповнення з іншими не дає повної завантаженості. Рішення – об'єднувати більш як два мікросервіси.

Підсумовуючи всі розв'язки вищезгаданих випадків, бачимо, що для групи мікросервісів будуть виділені сервери, а інші випадки слід звести до стандартних мікросервісів, які можуть знайти собі доповнення. Також, зважаючи на малий розмір і час застосування мікросервісів, що залишились, варто користуватись іншим алгоритмом їхнього збору (пункт 15).

13. **Розділення базового навантаження й екстремумів для одиноких особин.** Розглянемо випадок 12.2, коли існують мікросервіси з не дуже великим навантаженням, меншим за $P_{boundary} - P_{boundary} \times \alpha_{boundary}$, але все ж залишились без пари. Вірогідно, екстремуми трапляються в нестандартний час, або ж кількість денних завантажень переважає нічні завантаження, тому пари не існує. Проте є вірогідність знайти пару, якщо "згладити" піки роботи такого мікросервісу, коли їх небагато.

Для визначення стандартного навантаження та піків часового ряду можна проаналізувати значення середнього та стандартного відхилень (Zieffler, & Catalysts for Change, 2019), що допоможе виявити екстремуми. Знайдемо середнє значення часового ряду:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

Визначимо стандартне відхилення:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}.$$

Екстремуми визначають як значення, що перевищують певну кількість стандартних відхилень від середнього. В цій задачі одного стандартного відхилення вже достатньо для такої оцінки, тобто, якщо значення виходять за межі $\mu \pm \sigma$, їх можна вважати екстремумами (аномаліями). Нас цікавлять тільки пікові значення, бо їх можна "зрізати", виділивши під них окремий мікросервіс. Для монолітів указана практика схожа: доведеться створювати дублювальний моноліт у вибраний час і виконувати розподіл навантаження (load balancing) між двома екземплярами.

Має сенс зрізання всіх значень, які йдуть підряд і перевищують середнє. Для наочності наведено приклад. На рис. 3 показано часовий ряд із вираженим піком, через який важко знайти доповнювальний мікросервіс. Цей пік потрібно зрізати, тоді графік буде рівніший, знайти доповнення буде ймовірніше.

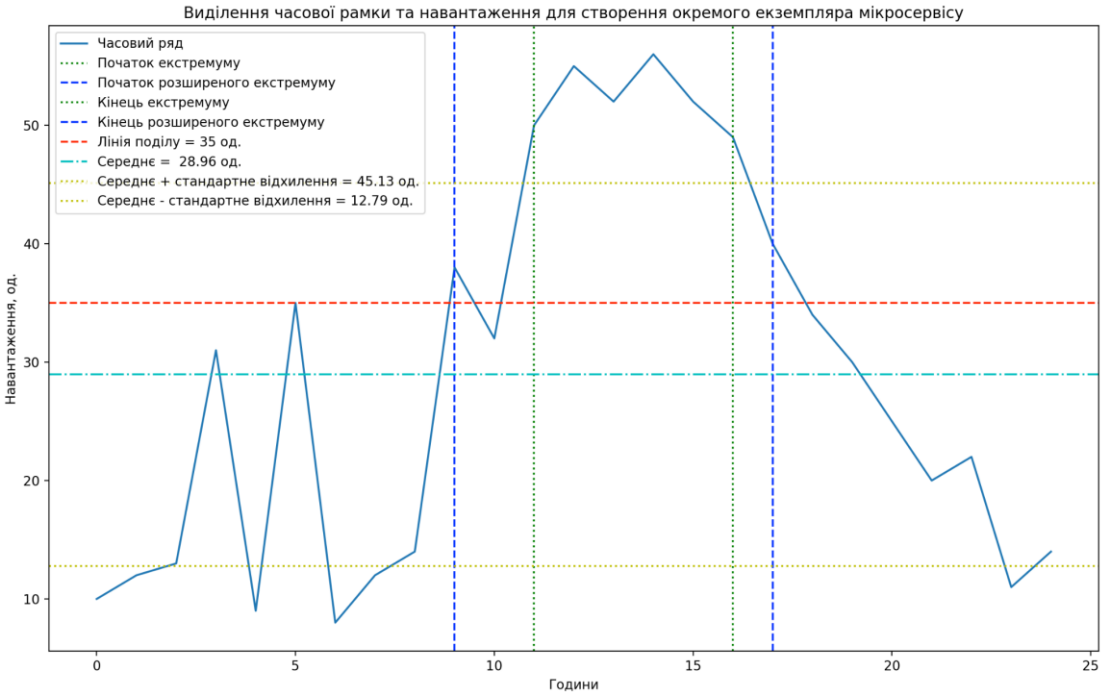

Рис. 3. Виділення навантаження мікропроцесора, яке варто запустити на іншому сервері. Стосовно моноліта – визначення часу для його додаткового запуску

За згаданим вище алгоритмом знайдуться пікові точки, що лежать вище верхньої жовтої лінії (з крапочок). Ці пікові значення позначено зеленими вертикальними лініями (з крапочок). Екстремуми, розміщені нижче нижньої жовтої лінії не розглядають. Купол графіка спостерігається довший час, ніж у верхньому квадраті, оточеному зеленими та жовтою лініями (з крапочок). Є сенс розширити цей проміжок у випадку, якщо він розташований вище середнього навантаження, позначеного голубою лінією (як крапка-тире). Для цього алгоритм проходить по всіх ближніх точках ліворуч і праворуч від піка, а також підбирає ті, що більші за середнє значення.

Відтак проводять пошук максимальної точки у часовому ряді, який не включає пік і близькі точки. Ця точка і буде навантаженням поділу мікросервісу. Наступним кроком буде відкидання всіх потенційно-пікових точок, що лежать між жовтою та голубою лініями, які менші за обраний новий максимум. Останній позначено червоною горизонтальною лінією (як тире). Розширений екстремум позначено синіми вертикальними лініями (тире).

На рис. 4 показано поділ мікросервісу на дві частини: до нової максимальної лінії та вище за неї.

Плануючи дії клауд-провайдера, має сенс заздалегідь запустити пікове навантаження на окремому сервері чи серверній групі, ввівши його в стан гарячого очікування (hot stand-by) (Levitin, Xing, & Dai, 2014).

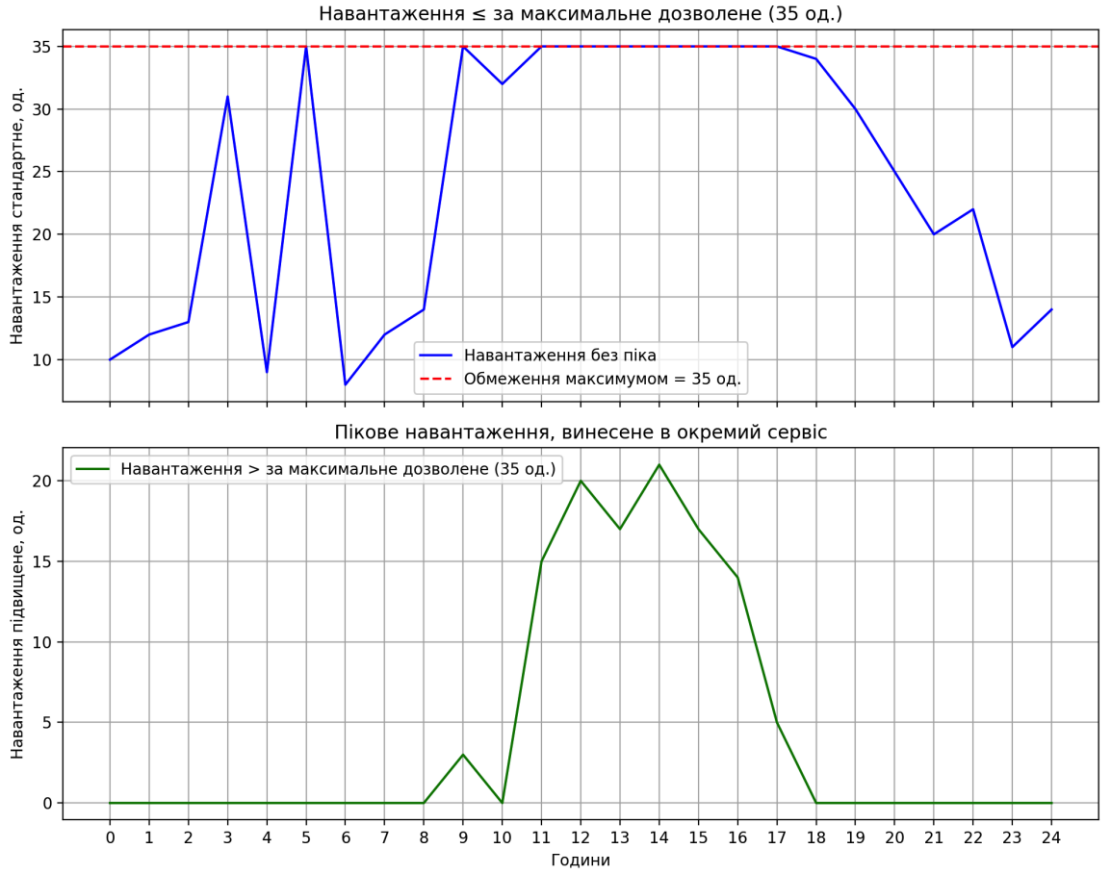

Рис. 4. Поділ мікросервісу на дві частини, екстремум виділено в окремий екземпляр

14. **Пошук доповнень для розділених мікропроцесорів**. Оскільки мікросервіси змінились із часу останнього пошуку доповнень, потрібно повторити крок 11.

15. **Комбінація малогабаритних мікросервісів.** Для решти мікросервісів є два наступні кроки, перший можна опустити:

- оскільки розмір мікросервісів малий, можна кластеризувати всі мікросервіси через KMeans, об'єднати по 2–3 мікросервіси з груп неперетину, у яких різні доповнювальні групи, після чого прогнати нові групи по кроку 11;
- решту мікросервісів, які залишаться, об'єднати, користуючись алгоритмом упакування множинного рюкзака із гнучким обмеженням цільової ваги (Cacchiani, Iori, Locatelli, & Martello, 2022).

Отже, всі мікропроцесори будуть згруповані, щоб обчислювальні потужності максимально не простоювали, а також щоб можна було підготувати потрібну машину для запуску обчислень на ній аби зменшити час розігріву, ввівши її в стан гарячого очікування (Levitin, Xing, & Dai, 2014).

16. **Налаштування під час роботи ПЗ на боці провайдера.** Після запуску система потребує продовження моніторингу. На цій стадії потрібно слідкувати за оновленням метрик і за якістю розподілу навантажень. Якщо виявлено, що протягом зазначеного часу серверна група використовує замалу кількість ресурсів, чи навпаки, потребує більше, то такі групи варто виокремити, скомбінувати з новими мікросервісами за нагоди й виконати пошук нових доповнень.

Результати

Під час алгоритму пошуку доповнювальних мікросервісів виконується кластеризація та екземпляри мікросервісів групуються у класи еквівалентності на основі подібності шаблона роботи. В межах груп екземпляри сортуються за

ISSN 2788-6603

амплітудою використання ресурсів, після чого групи, які визначаються як доповнювальні, перетинаються з метою пошуку конкретної пари комплементарних мікросервісів. У результаті, екземпляри зі значними відмінностями в шаблонах (протилежні шаблони навантаження) комбінуються з іншими, що мають подібні амплітуди, але протилежні фази, для максимального використання ресурсів. Комбінації з екземплярами, що мають низьку амплітуду, також можуть бути доречними для заповнення "дірок". У доповнювальних групах, алгоритм шукає перший екземпляр мікросервісу, що відповідає умовам доповнення (жадібний алгоритм). У разі виникнення труднощів знайти доповнення в першій найліпшій групі, зберігається статистика комбінацій з усіма екземплярами. Пошук продовжується до знаходження доповнень усім мікросервісам групи або поки не буде перебрано всі доповнювальні групи. Таких ітерацій проводять стільки, скільки будуть знаходитись пари, але "залишки" після пошуку комбінацій є очікуваними. Мікросервіси, що не знайшли доповнень додатково поділяють із використанням розширеного пошуку екстремумів за допомогою пошуку середнього та стандартного відхилень для збільшення їхніх шансів знайти компліменти. Кластеризацію та пошук проганяють до останньої успішної знахідки. Коли лишається невелика кількість мікросервісів, які не знайшли пари, комбінування відбувається за допомогою задачі про множинний рюкзак.

Результати роботи алгоритму – це скорочення кількості необхідних серверів до 15–20 %. Крім того, цей алгоритм дає можливість сформувати очікувану поведінку та ввести у стан розігріву додаткові мікросервіси, щоб користувачі ПЗ не відчували затримок у відповідях при їх різкому зростанні.

Відсоток економії серверів може варіюватись залежно від співвідношення локації хмарної технології до локації використання програмного забезпечення, а також від характеру розв'язуваних задач і збалансованості навантаження на хмарі, зокрема у денний і нічний час. Порівняно з поточним станом справ на хмарних ресурсах та Kubernetes, за потреби починає розгортатись новий екземпляр мікросервісу в момент, коли поточні метрики повідомлять про це (Скулиш, & Дмитренко, 2024; Дмитренко, & Скулиш, 2024b). Подібну ситуацію показано на рис. 4. Економія проявлятиметься у час, коли ресурс не повністю заповнений, тобто в години 8–11, 16–18, адже сервер мало завантажений у цей час, але, користуючись цим алгоритмом, він буде завантажений близько до повного.

Енергоефективність, яка буде наслідком економії серверних ресурсів, є однією з цілей ООН (United Nations Development Programme, 2024). Природними методами перероблення електроенергії у формат для вживання важко забезпечити потреби людства. Зменшуючи потреби в електроенергії, ми наближаємо кращу екологічну ситуацію у світі.

Дискусія і висновки

Наведено детальний опис алгоритму знаходження комплементарних мікросервісів, які за комбінування утворюють загальне навантаження на групу серверів, близьке до максимального. Цього можна досягти за допомогою різних технік поділу та поєднання мікросервісів, використовуючи вхідну та поточну статистику навантаження на канал, процесор або оперативну пам'ять.

Алгоритм містить нормалізацію вхідних результатів, кластеризацію методом K-Means, визначення доповнювальних груп, і на їхній основі – комплементарних пар мікросервісів, пропонує механізм поділу недоповнених мікросервісів на менші методом середнього та стандартного відхилень, а також передбачає комбінацію малогабаритних мікросервісів із більше ніж одним доповненням, користуючись методом упаковування множинного рюкзака. З метою візуалізації також запропоновано скористатись алгоритмами Z-Scale та FastDTW.

Внесок авторів: Олександра Дмитренко – розроблення методів і методології дослідження, опис результатів і написання висновків, огляд літературних джерел, збір емпіричних даних і проведення емпіричних досліджень; Марія Скулиш – поради щодо розроблення методів і методології дослідження й опису результату.

Список використаних джерел

Дмитренко О., & Скулиш, М. (2024b). Методи збору інформації та реалізації алгоритму доповнювальних навантажень. *Системи управління, навігації та зв'язку. Збірник наукових праць,* 4(78), Article 78. https://doi.org/10.26906/SUNZ.2024.4.056

Скулиш, М. А., & Дмитренко О. А. (2024). Метод організації мікросервісів на серверних групах Kubernetes. *Вчені записки Таврійського Національного Університету Імені В. І. Вернадського. Серія "Технічні науки"*, 1(5), 291–297. https://doi.org/10.32782/2663-5941/2024.5.1/41

Aspire Systems. (2023, June 22). Microservices Architecture: The Foundation of Cloud-Native Applications. *Software Engineering*. https://blog.aspiresys.com/software-product-engineering/microservices-architecture-the-foundation-of-cloud-native-applications/

Cacchiani, V., Iori, M., Locatelli, A., & Martello, S. (2022). Knapsack problems – An overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problems. *Computers & Operations Research*, 143, 105693. https://doi.org/10.1016/j.cor.2021.105693

Dmytrenko, O., & Skulysh, M. (2024a). Method of Grouping Complementary Microservices Using Fuzzy Lattice Theory. In E. Siemens (Eds). *International Conference on Applied Innovations in IT (ICAIIT)*, 12(1), 11–18. Anhalt University of Applied Sciences Bernburg / Koethen / Dessau. https://doi.org/10.25673/115636

Dmytrenko, O., Skulysh, M., & Globa, L. (2024). Microservice Complimentary Groups Determination Algorithm for the Effective Resource Usage. In I. Sinitsyn & P. Andon (Eds.), Proceedings of the 14th International Scientific and Practical Programming Conference (UkrPROG 2024) (Vol. 3806, pp. 180–201). CEUR. https://ceur-ws.org/Vol-3806/#S_55_Dmytrenko_Skulysh_Globa

Levitin, G., Xing, L., & Dai, Y. (2014). Cold vs. Hot standby mission operation cost minimization for 1-out-of-*N* systems. *European Journal of Operational Research*, 234(1), 155–162. https://doi.org/10.1016/j.ejor.2013.10.051

Ozdemir, S., & Susarla, D. (2018). Feature Engineering Made Easy. Packt Publishing. https://www.oreilly.com/library/view/feature-engineering-made/9781787287600/

Salvador, S., & Chan, P. (2004). Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intelligent Data Analysis*, 11, 80. https://cs.fit.edu/~pkc/papers/tdm04.pdf

Sharma, Y. (2023, September 26). *Key Strategies for Implementing AWS Network Load Balancer (AWS Community Builders).* DEV Community. https://dev.to/aws-builders/key-strategies-for-implementing-aws-network-load-balancer-35fc

Zieffler, A., & Catalysts for Change. (2019). Statistical Thinking: A Simulation Approach to Modeling Uncertainty (UM STAT 216 edition) (4.2).). In National Science Foundation (Eds.). MN: Catalyst Press. https://bookdown.org/frederick_peck/textbook/the-mean-and-standard-deviation.html MN: Catalyst Press. https://bookdown.org/frederick_peck/textbook/the-mean-and-standard-deviation.html

United Nations Development Programme. (2024). *Цілі сталого розвитку*. UNDP. https://www.undp.org/uk/ukraine/tsili-staloho-rozvytku

References

Aspire Systems. (2023, June 22). Microservices Architecture: The Foundation of Cloud-Native Applications. *Software Engineering*. https://blog.aspiresys.com/software-product-engineering/microservices-architecture-the-foundation-of-cloud-native-applications/

Cacchiani, V., Iori, M., Locatelli, A., & Martello, S. (2022). Knapsack problems – An overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problems. *Computers & Operations Research*,143, 105693. https://doi.org/10.1016/j.cor.2021.105693

Dmytrenko, O., & Skulysh, M. (2024a). Method of Grouping Complementary Microservices Using Fuzzy Lattice Theory. In E. Siemens (Eds). *International Conference on Applied Innovations in IT (ICAIIT)*, 12(1), 11–18. Anhalt University of Applied Sciences Bernburg / Koethen / Dessau. https://doi.org/10.25673/115636

Dmytrenko, O., & Skulysh, M. (2024b). Methods of data collection and implementation of accomplishing loads algorithm. Control, Navigation and Communication Systems. Academic Journal, 4(78), Article 78. https://doi.org/10.26906/SUNZ.2024.4.056

Dmytrenko, O., Skulysh, M., & Globa, L. (2024). Microservice Complimentary Groups Determination Algorithm for the Effective Resource Usage. In I. Sinitsyn & P. Andon (Eds.), Proceedings of the 14th International Scientific and Practical Programming Conference (UkrPROG 2024) (Vol. 3806, pp. 180–201). CEUR. https://ceur-ws.org/Vol-3806/#S_55_Dmytrenko_Skulysh_Globa

Levitin, G., Xing, L., & Dai, Y. (2014). Cold vs. Hot standby mission operation cost minimization for 1-out-of-N systems. European Journal of Operational Research, 234(1), 155–162. https://doi.org/10.1016/j.ejor.2013.10.051

Ozdemir, S., & Susarla, D. (2018). Feature Engineering Made Easy. Packt Publishing. https://www.oreilly.com/library/view/feature-engineering-made/9781787287600/

Salvador, S., & Chan, P. (2004). Toward Accurate Dynamic Time Warping in Linear Time and Space. In Intelligent Data Analysis (Vol. 11, p. 80). https://cs.fit.edu/~pkc/papers/tdm04.pdf

Sharma, Y. (2023, September 26). Key Strategies for Implementing AWS Network Load Balancer [AWS Community Builders]. DEV Community. https://dev.to/aws-builders/key-strategies-for-implementing-aws-network-load-balancer-35fc

Skulysh, M. A., & Dmytrenko, O. A. (2024). A Method of Organisation of Microservices on Kubernetes Server Groups. Scientific notes of Taurida National V. I. Vernadsky University. Series: Technical Sciences, 1(5), 291–297. https://doi.org/10.32782/2663-5941/2024.5.1/41

Zieffler, A., & Catalysts for Change. (2019). Statistical Thinking: A Simulation Approach to Modeling Uncertainty (UM STAT 216 edition) (4.2).). In National Science Foundation (Eds.). MN: Catalyst Press. https://bookdown.org/frederick_peck/textbook/the-mean-and-standard-deviation.html MN: Catalyst Press. https://bookdown.org/frederick_peck/textbook/the-mean-and-standard-deviation.html

United Nations Development Programme. (2024). Goals of sustainable development. UNDP. https://www.undp.org/uk/ukraine/tsili-staloho-rozvytku

Отримано редакцією журналу / Received: 21.10.24
Прорецензовано / Revised: 15.11.24
Схвалено до друку / Accepted: 20.11.24

Oleksandra DMYTRENKO, PhD Student, Assist.
ORCID ID: 0009-0009-4785-4226
e-mail: o_dmytrenko@lll.kpi.ua
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

Mariia SKULYSH, DSc (Engin.), Prof.
ORCID ID: 0000-0002-5141-1382
e-mail: mskulysh@gmail.com
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

MICROSERVICE GROUPING FOR UNDERUSED RESOURCE USAGE

Background. ***The task of optimizing cloud system resource usage is of high priority, as this technology has become widely adopted and easily accessible. Efficient services are common to be created to beastly adapt to different types of products, e.g. serverless technologies and proprietary databases, and cloud-optimized software. Performance testing of cloud applications has become significantly easier with the emergence of Docker and Kubernetes, which has further increased the demand for cloud resources. Since a considerable amount of resources is spent on the development and maintenance of cloud infrastructure, it is essential to understand how these resources can be optimized.***

Methods. ***The basic idea is to reorganize cloud infrastructure so that the software is distributed into the server groups, not the servers in a complementary manner, meaning that the hardware load will be close to the full load. To form complementary groups, methods of clusterization (K-Means), multiple knack's problem, sorting, binary search, shuffling and division based on mean and standard deviation are used. Additionally, FastDTW and Z-scale algorithms are proposed for visualization.***

Results. ***The article examines the key characteristics of computer systems that can be measured and influenced. These characteristics include transmission channel bandwidth, delay-based performance evaluation, the amount of RAM and permanent storage, processing power, and the number of cores. An algorithm is proposed to identify complementary microservice instances that could efficiently utilize server resources. Initially, microservice instances are classified by their capacity, considering small instances as a unit of capacity. Through analysis, the microservice instances are grouped into equivalence classes based on similarity. The instances are then sorted by the amplitude of resource usage. Ideally, instances with significant differences in load are combined with others that have similar amplitudes but opposite phases to maximize resource utilization. Combinations with instances that have low amplitude may also be appropriate. Within opposite classes, which differ in the activity phases of microprocessor instances, the algorithm searches for the first microservice instance that meets the conditions. To achieve this, statistics of combinations with all instances are stored until the first successful combination is found. Otherwise, the search continues till the least possible combination. The leftovers after the combinations search are expected. They are additionally divided using the extended extremums search with the mean and standard deviation search to increase their chances to find compliments. Finally, the small amount of microservices that didn't find a match is combined using multiple knack's problem.***

Conclusions. ***The conclusions of the article suggest an algorithm that will optimize the use of cloud system resources by effectively distributing the load between different microservices. Optimization means to maximize the use of server resources by filling its free time with other microservice and so reduce server downtime. This, in turn, will contribute to improving general cloud productivity and reducing maintenance costs of cloud infrastructures. This algorithm can be useful for cloud service providers and organizations that use cloud environments to deploy their applications.***

Keywords: ***microservice grouping, CPU efficiency, energy consumption optimization, cloud systems, microservice architecture, technology cost reduction.***

Автори заявляють про відсутність конфлікту інтересів. Спонсори не брали участі в розробленні дослідження; у зборі, аналізі чи інтерпретації даних; у написанні рукопису; в рішенні про публікацію результатів.

The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; in the decision to publish the results.

ISSN 2788-6603