УДК 004.7
DOI https://doi.org/10.32782/2663-5941/2024.5.1/41

Скулиш М.А.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Дмитренко О.А.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

МЕТОД ОРГАНІЗАЦІЇ МІКРОСЕРВІСІВ НА СЕРВЕРНИХ ГРУПАХ KUBERNETES

У статті розглянуто сучасні методи оптимізації розподілу обчислювальних ресурсів у хмарних середовищах шляхом організації мікросервісів. Основна увага зосереджена на декомпозиції монолітних додатків та мікросервісів з високими вимогами до ресурсів, що дозволяє підвищити ефективність використання серверних потужностей. Автори пропонують метод кластеризації мікросервісів з використанням алгоритмів KMeans та DBSCAN, що дозволяє ефективніше групувати сервіси за їхньою поведінкою та навантаженням. Це дає можливість оптимізувати процес балансування навантаження та забезпечити більш стабільну роботу системи при пікових навантаженнях. Зокрема, у статті розглядається як використовувати запропонований підхід в роботі оркестраторів контейнерів, таких як Kubernetes, для управління мікросервісами. Автори описують концепцію використання балансувальників "sidecar", що розгортаються поруч із кожним контейнером мікросервісу для маршрутизації запитів. Такий підхід дозволяє уникнути "єдиного вузла відмови" та забезпечує більш рівномірний розподіл запитів між вузлами. Особлива увага приділяється енергоефективності та оптимальному використанню ресурсів. У статті розглядаються методи поділу мікросервісів з високими вимогами до ресурсів на менші компоненти, що дозволяє гнучкіше керувати системою при змінних навантаженнях. Така декомпозиція знижує ризики перевантаження системи і підвищує її загальну продуктивність. Така робота є важливою для забезпечення стабільності сучасних хмарних систем, зокрема у контексті швидкого розвитку технологій мікросервісної архітектури та контейнеризації.

Ключові слова: *доповнення мікросервісів, хмарні обчислення, кластеризація мікросервісів, енергоефективність, оптимальне завантаження ресурсів, Kubernetes.*

Постановка проблеми. Архітектура мікросервісів стала пануючою в хмарному середовищі. Вона спрощує розробку додатків, розбиваючи монолітні застосунки на управляючі мікросервіси, які можна розробляти та розгортати незалежно від усього цілого. Однак перехід від монолітної або простої багаторівневої архітектури до розподіленої мікросервісної призводить до нових викликів через більш складну топологію додатків. Особливою проблемою при автоматичному керуванні продуктивністю мікросервісів є те, що, оскільки кожен компонент сервісу масштабується незалежно вгору та вниз, це може легко призвести до проблем нерівномірного навантаження на загальні послуги зв'язку, які використовуються кількома компонентами. Традиційні алгоритми балансування навантаження були розроблені для централізованих балансувальників, розташованих між групою клієнтів та фермою серверів. Проте ці алгоритми не ефективно переносяться на розподілену мікросервісну архітектуру, де балансувальники навантаження розгортанні на боці клієнта.

Аналіз останніх досліджень і публікацій. Останні роки дослідники активно працюють над розвитком архітектур мікросервісів і їхньої оптимізації в хмарних середовищах. У статті [1] О. Аль-Дебагі та П. Мартінек порівнюють мікросервісні та монолітні архітектури, акцентуючи увагу на перевагах розподілених систем, зокрема, на можливості незалежного масштабування компонентів і їхньої гнучкості. Це дозволяє значно покращити ефективність роботи програмних систем у різних середовищах. У публікаціях [2, 7] розглядаються підходи до визначення груп мікросервісів для ефективного використання обчислювальних ресурсів, що дозволяє підвищити продуктивність систем та оптимізувати балансування навантаження. Їхні роботи надають рекомендації щодо організації мікросервісів у Kubernetes, зокрема з використанням методів кластеризації

KMeans і DBSCAN для групування за схожими характеристиками навантаження. Інші дослідження, наприклад [3], пропонують стратегії для підвищення відмовостійкості мікросервісів у IoT середовищах, а також методи для забезпечення надійності при змінних умовах навантаження. Отже, сучасні дослідження спрямовані на оптимізацію використання ресурсів у хмарних системах за допомогою архітектур мікросервісів, а також на впровадження більш гнучких і стійких до помилок рішень для забезпечення ефективної роботи додатків.

Постановка завдання. Метою статті є дослідження методів організації мікросервісів у середовищі Kubernetes для ефективнішого розподілу обчислювальних ресурсів. Це передбачає оптимізацію використання потужностей серверів через декомпозицію монолітних додатків і мікросервісів з високими вимогами до ресурсів. А також аналіз технік групування мікросервісів за моделями навантаження та застосування алгоритмів кластеризації для поліпшення продуктивності системи.

Виклад основного матеріалу.

Інфраструктура для обслуговування мікросервісів

Мікросервіси стали все більш популярними завдяки різноманітним перевагам, таким як легкість розгортання, постійна інтеграція, незалежний розвиток та інші. У більшості сучасних сценаріїв мікросервіси розгортаються як контейнери в кластерах, керованих оркестратором, таким як Kubernetes. Використання шаблонів мікросерсів, пов'язаний з контейнерними кластерами, стало дуже популярним. Що в свою чергу призвело до зменшення ролі централізованих балансувальників навантаження на одному вузлі. Замість цього, балансувальники навантаження на боці клієнта розгортаються поруч із кожним контейнером служби як "sidecar", як показано на рис. 1. Перевага використання цього принципу полягає в тому, що балансувальник вилучається як єдиний точковий вузол відмов або проблем з продуктивністю.

У розгортанні мікросервісів загальним явищем є те, що фонові сервіси можуть бути використані кількома вихідними компонентами, кожен з яких може бути реплікованим. У такому сценарії кожен вихідний вузол надсилає лише невелику частину від загальної кількості запитів, які отримує кожен вихідний вузол. Це призводить до розходження між фактичним завантаженням вихідних вузлів та оцінкою того завантаження, яке мають вихідні вузли. У результаті продуктивність додатку може

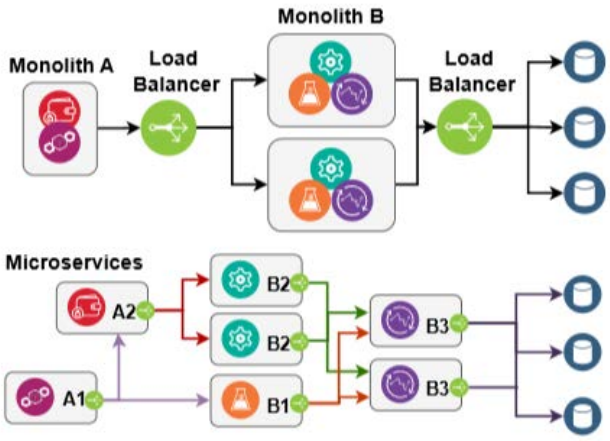

Рис. 1. Багаторівневий додаток, побудований на основі монолітних сервісів (зверху), може бути розкладений на компоненти мікросервісів (знизу), що потенційно поліпшує практики розробки, але ускладнює топологію додатку. Балансувальники навантаження "sidecar" (зелені круги) розгортані поруч із кожним компонентом мікросервісу для маршрутизації запитів до вузлів обслуговування (нижні вузли)

швидко погіршитися через неправильні рішення, прийняті такими «локальними» алгоритмами.

Поділ мікросервісів з високими вимогами до ресурсів

Архітектура мікросервісів стає все більш популярною, оскільки дозволяє розділити монолітний додаток на окремі менші сервіси. Це сприяє скороченню часу розробки, прискорює цикли впровадження, забезпечує можливість використання різних технологічних стеків для окремих компонентів додатку, а також спрощує заміну окремих частин і дозволяє безперервно інтегрувати зміни без впливу на роботу всієї системи.

Для ефективного керування мікросервісами необхідно контролювати його розміри, тобо вимоги до обчислювальних ресурсів. Мікросервіс, який має високі вимоги до ресурсів, потребує розділення. Для вирішення задачі ефективного обслуговування мікросерсів пропонується виконати процедури групуванням та пошуком схожих та взаємодоповнюючих мікросервісів. Однак це не дасть своїх результатів без необхідні фільтрації мікросервісів, які можуть не підійти. Це мікросервіси з надзвичайно високими вимогами до ресурсів. Позначимо як момент часу, коли навантаження досягає максимального значення. Тоді критерій для поділу виглядає наступним чином:

$$P\mathrm{t}_{max} + \delta > P\mathrm{boundary} \qquad (1)$$

Ці мікросервіси повинні бути розділені на менші частини, якщо їх можна розбити на частини.

Можливо, це буде монолітний додаток, оскільки ідея мікросервісів пов'язана з зазвичай невеликими завданнями, які відокремлені від інших [1], тому вони не повинні займати багато місця. Більше інформації про поділ мікросервісів та використання меж можна знайти в [2]. Такий поділ може допомогти передбачити час запуску мікросервісів і підготувати додаткові сервери, встановивши на них передстартові дані та записавши останню актуальну інформацію, наприклад, про холодний, теплий або гарячий стан очікування [3].

При декомпозиції монолітного додатку на менші компоненти доцільно базувати сегментацію на звичайному або середньому навантаженні, з яким працює система. Для періодів, коли навантаження перевищує середнє, ефективно передбачити додаткові екземпляри моноліту, які налаштовані на обробку підвищеного попиту. Ця стратегія гарантує, що система залишатиметься швидкою та стабільною під час пікових навантажень, водночас оптимізуючи використання ресурсів під час нормальної роботи. На рис. 2 наведена візуалізації чього процесу.

Існують ситуації, коли створення додаткових екземплярів програми може бути недоцільним або невигідним. Наприклад, якщо додаток починає послідовно отримувати незвично великі повідомлення – замість невеликих, частих – це може призвести до перевантаження ресурсів каналу. Це слід розглядати як винятковий випадок. Крім того, можна переробити код мікросірвіса так, щоб керування передачею даних здійснювалос більш ефективно, тим самим зменшуючи потребу в додаткових екземплярах.

Для цієї частини алгоритму будуть наступні вхідні значення:

«canChunk» – ця характеристика вказує на те, чи можна розділити мікросервіс на менші, більш керовані частини, та максимально можливі значення, на основі яких буде відбуватися відсікання надмірних навантажень:

– P_{boundary}, RAM
– P_{boundary}, CPU
– P_{boundary}, Channel

Алгоритм поділу наступний:

1. Визначити мікросервіси з вимогами до ресурсів, що перевищують P_{boundary}, RAM, P_{boundary}, CPU або P_{boundary}, Channel.
2. Перевірити характеристику «canChunk».
3. Якщо (1) виконується, розділити мікросервіси з високими вимогами (опис та приклад наведено нижче).
4. Сформувати окрему групу «Високовимогливих мікросервісів» для тих, які не можна розбити на частини, але які перевищують межі. Елементи з цієї групи займатимуть окремі великі ресурси і не братимуть участі в наступному алгоритмі.

Поділ, згаданий у 3-му пункті, відбувається, якщо мікросервіс або моноліт можна і потрібно розбити на частини, він розбивається на менші частини з вимогами до ресурсів нижче максимальних порогів. Такий поділ здійснюється лише тоді, коли конкретна характеристика перевищує своє мінімальне значення на певний відсоток, відомий як «Δ» (дельта), а їх сума є нижчою за поріг. Δ являє собою додатковий відсоток, який слід додати до мінімального значення характеристики, що викликає поділ.

Нехай – мінімальний поріг для характеристики, – відсоток збільшення від 0% до 100%, – максимальний поріг. Ця формула повинна також включати невеликий відсоток варіацій δ, які

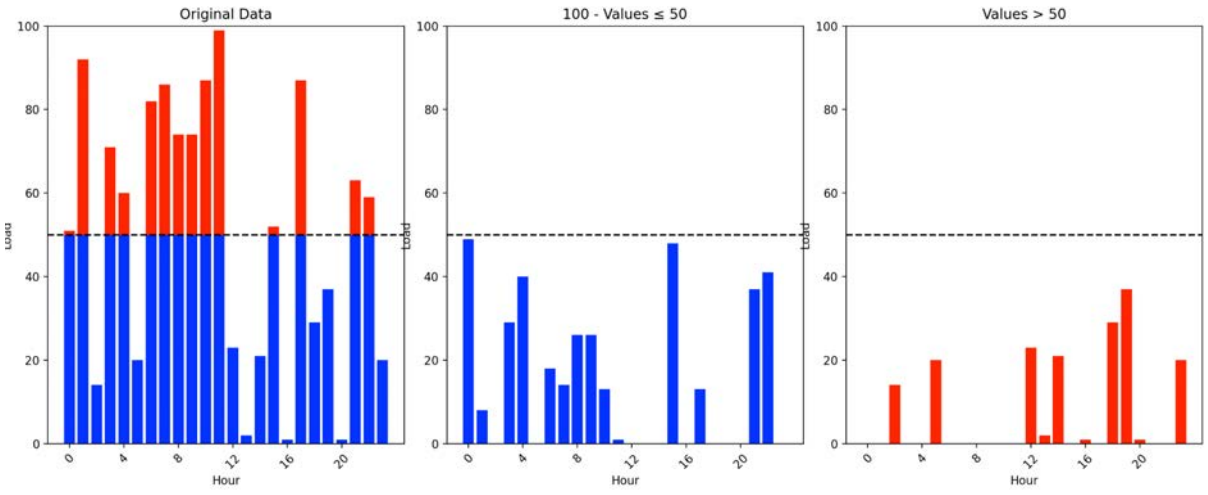

Рис. 2. Поділ високонавантаженого мікросервісу на 2 екземпляри

можуть відбуватися щодня. Цей параметр надається разом з початковими метриками. Алгоритм працює для всіх критеріїв однаково, тому формули і приклади написані без уточнення, для якої саме характеристики вони взяті. Поділ має відбуватися, якщо характеристика перевищує :

$$P_{chunk} = P_{tmin} + \Delta P_{tmin} + \delta < P_{boundary}, \quad (2)$$

де

P_{chunk} – поріг для початку поділу.

– момент з мінімальним значенням для певної характеристики.

Δ – відсоток збільшення, необхідний для поділу.

– максимальне порогове значення.

Наприклад, якщо Δ = 30%, δ = 5%, P_{tmin} = 10 і $P_{boundary}$ = 20. Підставивши значення у формулу, отримаємо наступне:

$$P_{chunk}=10+0.30\times10+0.05\times10=10+3+0.5=13.5$$

Оскільки 13,5 менше граничного значення 20, то мікропослуга повинна бути розділена, якщо характеристика перевищує 13,5 одиниць. Це гарантує, що мікросервіси ефективно розділяються тільки тоді, коли це необхідно, оптимізуючи використання ресурсів у хмарному середовищі. Поділ все одно відбуватиметься за граничним значенням, а не за Δ ідсотків.

Для визначення кількості частин, на які слід розділити мікросервіс, ми використовуємо формулу (3). Ця формула враховує суму максимального навантаження P_{max} та невеликий відсоток для врахування добових коливань δ, ділить цю суму на граничне значення $P_{boundary}$ та округляє до найближчого цілого числа, яке позначено функцією «підлога-стеля».

$$m = \left\lfloor \frac{P_{max} + \delta}{P_{boundary}} \right\rceil \quad (3)$$

Отримане в результаті останнє навантаження на мікропроцесор може виявитися занадто малим і ним можна знехтувати. Критерії повинні бути специфічними для хмарних та інших обчислень. Ми пропонуємо не створювати останній блок, якщо залишок не перевищує δ. Цей параметр буде використано в інших формулах і його слід враховувати при підрахунку загального навантаження групи. У формулах це виглядає наступним чином:

якщо $\left(\left\lceil \frac{P_{max} + \delta}{P_{boundary}} \right\rceil - \frac{P_{max} + \delta}{P_{boundary}} \right) \leq \delta$,

то $m = \left\lfloor \frac{P_{max} + \delta}{P_{boundary}} \right\rfloor$

інакше $m = \left\lceil \frac{P_{max} + \delta}{P_{boundary}} \right\rceil \quad (4)$

Якщо P_{max} є 50, δ є 8% або 0,08, що становить 50 * 0,08 = 4, і $P_{boundary}$ є 20, результати застосування (3) і (4) розраховуються наступним чином:

$$m = \left\lfloor \frac{50 + 4}{20} \right\rceil = \left\lfloor \frac{54}{20} \right\rceil = \lfloor 2.7 \rceil$$

якщо 3–2.7=0.7 значення < 0.08, то округляємо в більшу сторону |=3.

Мікросервіси та Sidecars

Мікросервіси зазвичай запускаються в контейнерах з використанням оркестраційних фреймворків, таких як Kubernetes. Якщо мікросервіси є частинами розподіленого монолітного сервісу, то системи контейнерної оркестрації йдуть далі, дозволяючи кожному мікросервісу розміщуватися в кількох контейнерах. Наприклад, один контейнер може містити бізнес-логіку додатку, а інші – виконувати функції моніторингу або балансування навантаження. Такі додаткові контейнери часто називають "sidecars", оскільки вони розгортаються поруч із основним контейнером і обробляють вхідні або вихідні запити. Група контейнерів, що включає як основні, так і допоміжні контейнери, які разом утворюють логічний сервіс, об'єднується в єдиний простір імен, відомий як "pod" у Kubernetes.

Оскільки кожен "pod" може бути репліковий для масштабування мікросервісів, для маршрутизації запитів до відповідних вузлів потрібні балансувальники навантаження. Можливість легко інтегрувати функціональні компоненти дозволила замінити централізовані балансувальники навантаження розподіленими балансувальниками "sidecar", які розгортаються разом із кожним "pod". Кожен "проксі-sidecar" відповідає за балансування вихідних запитів мікросервісу через кілька реплік у ланцюжку. Це робить систему більш масштабованою, хоча і без глобального огляду, як у централізованих рішеннях.

Групування на основі алгоритмів кластеризації KMeans та DBSCAN

Ефективне управління ресурсами в хмарних обчисленнях передбачає кілька етапів підготовки мікросервісів, групування на основі моделей використання ресурсів, а також пошук взаємодоповнюючих мікросервісів всередині цих груп. Коли група сформована, її можна розгортати. Такий підхід підвищує продуктивність та ефективність використання ресурсів. Крім того, поєднання в групи основних мікросервісів та “sidecars”, дозволить більш ефективно організовувати завантаженість подів оркестратора (Kubernetes’ pods).

Для групування мікросервісів за схожою поведінкою можна використовувати алгоритми клас-

теризації, такі як KMeans і DBSCAN (Density-Based Spatial Clustering of Applications with Noise). Об'єкти в одній групі, яка називається кластером, більш схожі один на одного, ніж на об'єкти в інших групах. KMeans і DBSCAN – це два широко використовувані алгоритми кластеризації, кожен з яких має свої особливості. У цьому розділі наведено детальне порівняння цих двох методів, зосереджено увагу на їхніх основних принципах, перевагах, обмеженнях та випадках використання.

Алгоритм DBSCAN ідентифікує кластери на основі щільності точок. Він може знаходити кластери довільної форми і може працювати з шумом. У цьому завданні очікується, що точки даних утворюють переважно кругові кластери, тому кластеризація за методом K-середніх виявляється більш підходящим вибором, ніж DBSCAN. Нижче описано алгоритми і показано, чому K-Means краще підходить для цього сценарію:

1. Очікується, що точки даних мають радіус кола. K-середнє краще виявляє сферичні або кругові кластери, що добре узгоджується з нашим очікуваним розподілом даних. DBSCAN, хоча і здатний виявляти кластери довільної форми, може надмірно ускладнити аналіз для наших круглих кластерів.

2. Для подальшого пошуку компліментів потрібні центроїдні точки як база, для якої шукатимуться компліменти. K-Means видає явні центроїди кластерів, що є цінним.

3. Обробка контурів не передбачається, оскільки вони мають бути відфільтровані на попередньому етапі. Здатність DBSCAN ідентифікувати точки шуму може бути корисною в деяких сценаріях, однак, завдання вимагає включення точок контуру до кластерів, як вони є. K-Means природним чином об'єднує всі точки в кластери, що є корисним, оскільки контури не слід розглядати як шум. Цей крок попередньої обробки підвищує ефективність K-Means для нашої задачі.

4. Під час ініціалізації нової компліментарної структури, згідно з ідеєю, описаною в статті, може виникнути потреба в обробці великої кількості мікросервісів. Саме тому обчислювальна ефективність є важливим моментом, який слід враховувати. K-Means зазвичай пропонує кращу масштабованість для великих наборів даних. У сценаріях, де швидкість обробки має вирішальне значення, K-Means може забезпечити швидші результати, особливо якщо кількість кластерів відома або може бути оцінена.

KMeans кластеризує дані в групи, мінімізуючи дисперсію всередині кожного кластера. Цільова функція полягає в тому, щоб знайти аргумент – центроїд, при якому функція відстані до точки матиме мінімальне значення. Іншими словами, найближчий центроїд повинен притягувати точку до свого кластера:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2 \qquad (5)$$

де

S – множина кластерів.

μ_i – центроїд кластера.

Алгоритм ітераційно оновлює центроїди і перепризначає точки до найближчих центроїдів до збіжності.

Існує також інший метод, який можна розглянути – косинусна подібність. Він вимірює косинус кута між двома векторами cos(θ), показуючи, наскільки вони схожі, незалежно від величини. Цей метод працював би добре, якби вектори були багатовимірними в реальності. Вектори в задачі насправді є часовими рядами. Ангели в різних частинах одного часового ряду є суперечливими. Метод косинусоїдальної подібності працює скоріше з лінією, а не з кривою, тому не підходить у цьому випадку.

Приклад групування

Щоб зробити приклад наближеним до реального, для кожного типу з п'яти поведінок було згенеровано 24 мікросервіси, які виконують одна одну. Поведінки з кількістю згенерованих мікропроцесорів представлені нижче:

– 2 мікросервіси зі стабільним навантаженням для.

4 мікросервіси з піковим навантаженням в робочий час (з 9:00 до 17:00).

2 мікросервіси з піковим навантаженням після опівночі та до 6 ранку.

2 мікросервіси з піковим навантаженням з 17:00 до ночі.

2 мікросервіси з випадковими піками.

– Доповнення для кожного з цих патернів (навантаження в протилежний час).

Кластеризація за допомогою алгоритму KMeans. Аналіз головних компонент (PCA) був використаний для векторів і центроїдів, щоб зробити зображення чітким і зрозумілим [4]. Алгоритм використовується лише для графічного представлення для кращого розуміння якості роботи алгоритму KMeans, тому в роботі він не описується. Більш детальну інформацію про нього можна знайти тут [5], [6]. Він зводить 24-вимірні дані до 2-вимірних, що дозволяє візуалізувати багатовимірні дані у вигляді двовимір-

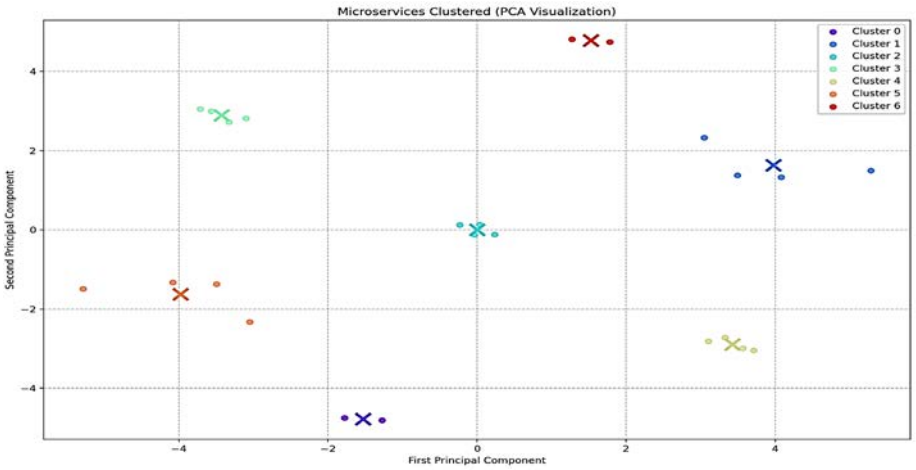

Рис. 3. 2D векторно-центроїдна візуалізація за алгоритмом KMeans після аналізу головних компонент. X – центроїди, точки – 24-D вектори, перетворені в 2-D вектори за допомогою PCA. Колір вказує на приналежність до певних центроїдів

ного графіка. Проте, слід зазначити, що частина інформації неминуче втрачається в процесі зменшення розмірності.

На діаграмі розсіювання на рисунку 3 кожна точка представляє мікропослугу. Центроїди позначені великими маркерами «X». Точки одного кольору утворюють кластер. Координати x та y кожної точки є першою та другою головними компонентами відповідно. Така візуалізація дозволяє спостерігати, наскільки добре розділені кластери і чи є в даних якісь чіткі угруповання.

Висновки. У статті було запропоновано та досліджено методи організації та групування мікросервісів у серверні групи Kubernetes. Основні результати роботи полягають у наступному:

1. Ефективне використання ресурсів: Запропонований підхід з декомпозицією мікросервісів дозволяє ефективно управляти високонавантаженими компонентами системи, оптимізуючи їх розподіл між серверами. Це підвищує загальну продуктивність системи та зменшує ризики перевантаження окремих ресурсів.

2. Декомпозиція мікросервісів: Процедура поділу мікросервісів з високими вимогами до ресурсів на менші компоненти дозволяє краще розподілити навантаження в середовищі Kubernetes. Це забезпечує більш гнучке управління ресурсами під час пікових навантажень.

3. Балансування навантаження: Використання алгоритмів балансування, таких як "sidecar", дозволяє більш рівномірно розподіляти запити між мікросервісами, що зменшує ймовірність виникнення «точок відмови» та покращує надійність системи.

4. Застосування алгоритмів кластеризації: Алгоритми кластеризації, такі як KMeans і DBSCAN, допомагають ефективно групувати мікросервіси за їхньою поведінкою, що дозволяє краще адаптуватися до різних сценаріїв використання ресурсів і підвищити ефективність роботи системи.

Таким чином, запропоновані методи та алгоритми підвищують гнучкість і масштабованість систем на основі мікросервісів у хмарному середовищі Kubernetes, забезпечуючи стабільну та ефективну роботу навіть при змінних навантаженнях.

Список литературы:

1. O. Al-Debagy and P. Martinek, "A Comparative Review of Microservices and Monolithic Architectures," in *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, Nov. 2018, pp. 000149–000154. doi: 10.1109/CINTI.2018.8928192.

2. O. Dmytrenko and M. Skulysh, "Determining microprocessor groups for efficient utilization of processor capacities" *Problems of programing Forthcom.*, vol. 1, no. №2-3, Aug. 2024, [Online]. Available: https://docs.google.com/document/d/1ocqA-e7d-OAibFVGiEE2UcxZRKgYNn_yPX6e94dHwYM/edit?usp=sharing

3. 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.

4. I. T. Jolliffe, Ed., "Graphical Representation of Data Using Principal Components," in *Principal Component Analysis*, New York, NY: Springer, 2002, pp. 78–110. doi: 10.1007/0-387-22440-8_5.

5. I. T. Jolliffe, Ed., “Principal Components as a Small Number of Interpretable Variables: Some Examples,” in *Principal Component Analysis*, New York, NY: Springer, 2002, pp. 63–77. doi: 10.1007/0-387-22440-8_4.

6. I. T. Jolliffe, Ed., “Choosing a Subset of Principal Components or Variables,” in *Principal Component Analysis*, New York, NY: Springer, 2002, pp. 111–149. doi: 10.1007/0-387-22440-8_6.

7. 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.

8. M. Skulysh and S. Sulima, "Management of multiple stage queuing systems," The Experience of Designing and Application of CAD Systems in Microelectronics, Lviv, Ukraine, 2015, pp. 431–433, doi: 10.1109/CADSM.2015.7230895.

Skulysh M.A., Dmytrenko O.A. A METHOD OF ORGANISATION OF MICROSERVICES ON KUBERNETES SERVER GROUPS

The article discusses modern methods for optimizing the distribution of computing resources in cloud environments through the organization of microservices. The main focus is on the decomposition of monolithic applications and resource-intensive microservices, which helps to improve the efficiency of server capacity utilization. The authors propose a method for clustering microservices using KMeans and DBSCAN algorithms, which enables more effective grouping of services based on their behavior and load. This allows for optimizing the load balancing process and ensuring a more stable system operation during peak loads. In particular, the article explores how the proposed approach can be applied in container orchestrators like Kubernetes to manage microservices. The authors describe the concept of using "sidecar" load balancers deployed next to each microservice container for request routing. This approach avoids a "single point of failure" and ensures a more even distribution of requests between nodes. Special attention is given to energy efficiency and optimal resource utilization. The article discusses methods for splitting resource-intensive microservices into smaller components, allowing for more flexible system management under changing loads. Such decomposition reduces the risk of system overload and increases its overall performance. This work is essential for ensuring the stability of modern cloud systems, especially in the context of the rapid development of microservice architecture and containerization technologies.

Key words: *Microservice augmentation, cloud computing, microservice clustering, energy efficiency, optimal resource utilisation, Kubernetes.*