УДК 004.78

DOI: 10.18372/2073-4751.80.19787

Артамонов Є.Б.,
orcid.org/0000-0002-9875-7372,
e-mail: eart@kai.edu.ua,
Коцюр А.Б.,
orcid.org/0000-0003-4404-1966,
e-mail: anatolii.kotsiur@npp.nau.edu.ua,
Крант Д.В.,
orcid.org/0000-0003-4601-2827,
e-mail: danbiil988@gmail.com,
Радченко К.М.,
orcid.org/0009-0009-8056-6293,
e-mail: radchenko.kn@gmail.com

ПІДХІД ДО ОПТИМІЗАЦІЇ МОДЕЛІ РОЗГОРТАННЯ МІКРОСЕРВІСІВ В СИЛЬНОНАВАНТАЖЕНОМУ СЕРЕДОВИЩІ

Державний університет «Київський авіаційний інститут»

Вступ

З великим обсягом трафіку, необхідністю швидкої обробки даних та постійної доступності системи, розробники стикаються з викликами, що потребують нових архітектурних підходів. Мікросервісна архітектура стала відповіддю на ці виклики, розподіляючи функціональність додатка на окремі незалежні компоненти, які можна масштабувати й оновлювати автономно. Такий підхід значно підвищує стійкість додатків, оптимізує використання ресурсів і збільшує загальну продуктивність системи.

У традиційних монолітних додатках будь-які зміни або збільшення навантаження потребують перезапуску всього додатка, що створює додаткові затримки й підвищує ризик простоїв. Мікросервіси, навпаки, дозволяють легко адаптуватися до зростання обсягів запитів за допомогою незалежних сервісів, які працюють спільно та підтримують стабільну роботу системи навіть під час пікових навантажень. Однак впровадження мікросервісної архітектури вимагає ретельного проєктування моделей розгортання, оскільки стійкість ресурсів безпосередньо залежить від налаштувань управління, резервування та масштабування сервісів.

Однією з головних проблем розгортання мікросервісів є питання однорідності. Ідентичні екземпляри мікросервісів, розгорнуті на декількох вузлах, мають спільні вразливості, якими можуть скористатися зловмисники. Одна вразливість в базовому образі контейнера може вплинути на всі екземпляри, що призведе до загальносистемних збоїв. Це робить традиційні методи резервування неспроможними протистояти більш широким ризикам спільних вразливостей у середовищах високої доступності.

Відмовостійкість, здатність системи продовжувати функціонувати в умовах збоїв, стає особливо важливою в системах, заснованих на мікросервісах. Хоча надмірність є загальновживаною стратегією підвищення доступності, вона не усуває ризик, пов'язаний з одноманітністю екземплярів сервісів. Як наслідок, науковці дослідили концепцію різноманітності в програмних системах, яка передбачає розгортання різних версій або реалізацій сервісів, щоб зменшити вплив спільних вразливостей.

Дослідження в роботі зосереджуються на тестуванні підходів до розгортання мікросервісів, що здатні адаптуватися до змін у навантаженні та підвищувати надійність системи. Розглянуто

питання автоматизованого балансування навантаження, моніторингу, резервування та розподілу ресурсів між сервісами для забезпечення мінімальних простоїв і високої швидкості обробки запитів.

Аналіз досліджень

В останній час дослідження мікросервісних архітектур зосереджені на оптимізації стратегій розгортання для підвищення стійкості системи та ефективності використання ресурсів. Низка досліджень запропонувала рішення для вирішення типових проблем планування завдань, системних навантажень та відмовостійкості в мікросервісних середовищах.

В дослідженні [1] розглядаються стратегії автоматизованого розгортання мікросервісів з акцентом на розподілі ресурсів і управлінні життєвим циклом. В роботі показано, як ці стратегії можуть пом'якшити каскадні збої та оптимізувати використання ресурсів у великомасштабних розподілених системах.

В роботі [2] досліджувався перехід з монолітної на мікросервісну архітектуру системи, що дозволило витримувати великі навантаження на електронну навчальну систему.

В роботі [3] проводився аналіз управління ресурсами на хмарних платформах з використанням *Docker* та *Kubernetes*. Дослідження підкреслило накладні витрати, пов'язані з цими платформами. Результати дослідження показали, що оркестрація на базі *Kubernetes* скоротила час виконання задач за рахунок оптимізації використання ресурсів, що є необхідним для роботи додатків з високим попитом.

У дослідженні [4] було проаналізовано можливості *Kubernetes* для автомасштабування та управління ресурсами. Також продемонстровано переваги використання предиктивної аналітики для динамічного налаштування ресурсів на основі вимог робочого навантаження. Використання інструментів моніторингу, таких як *Prometheus* та *Grafana*, виявилося ефективним для оптимізації розподілу ресурсів у хмарних середовищах.

Дослідження [5] описує переваги модульності, узгодженості та відмовостійкості при створенні масштабованих і підтримуваних систем. Модульний дизайн не тільки спрощує процес розробки, але й покращує масштабованість системи та стійкість до відмов.

Планування задач і розподіл ресурсів для мікросервісів були розглянуті в роботі [6], де запропоновано модель динамічного планування задач, призначена для зменшення накладних витрат і підвищення продуктивності. Завдяки динамічному розподілу віртуальних машин і контейнеризації мікросервісів дослідження продемонструвало значне скорочення часу виконання і підвищення продуктивності в середовищах з високим навантаженням.

Роль мікросервісної архітектури у підвищенні кібербезпеки та операційної стійкості критично важливих систем розглянуто в роботі [7], де підкреслюються переваги мікросервісів, такі як відмовостійкість, масштабованість і адаптивність. Також досліджується інтеграція машинного навчання для покращення моніторингу безпеки та виявлення загроз у цих системах.

Аналіз сучасних практик проектування фреймворків мікросервісів був наведений у дослідженні [8]. Дослідження визначило головні принципи, такі як автономія сервісів, масштабованість та використання інструментів оркестрування контейнерів. Показано, що ці принципи підвищують надійність і маневреність системи, роблячи мікросервіси більш керованими та масштабованими.

Робота [9] досліджує ефективність використання алгоритму *LB-Diversity* для балансування навантаження для мікросервісів (саме цей алгоритм вирішено використовувати і в реальній системі, на базі якої проводиться дослідження).

Проведений аналіз підкреслює виняткову роль оптимізації стратегій розгортання для підвищення відмовостійкості, масштабованості та ефективності використання ресурсів мікросервісних архітектур. Ці вдосконалення сприяють

створенню надійних і підтримуваних мікросервісних систем, здатних працювати в середовищах з високим навантаженням.

Постановка проблеми

Відмовостійкість та оптимізація ресурсів систем на основі мікросервісів стикаються з кількома вагомими труднощами, передусім через такі проблеми, як відмова вузлів та спільні вразливості.

Загальні компоненти стають критичною точкою ризику, оскільки атака або збій, що впливають на один з них, можуть вивести з ладу декілька мікросервісів, призводячи до порушення роботи цілої системи. Особливо гостро ця проблема постає в системах, що працюють за принципом високої доступності. У таких системах мікросервіси, розгорнуті на різних вузлах, мають забезпечувати безперервність обслуговування великої кількості користувачів і витримувати значні навантаження без втрати продуктивності.

Висока доступність потребує надійного резервування та балансування навантаження, що додатково ускладнюється наявністю спільних вразливих компонентів. Невирішені проблеми з балансуванням призводять до нерівномірного розподілу ресурсів між мікросервісами, що знижує продуктивність і призводить до недовантаження або перевантаження окремих вузлів. Також додатковий виклик створює ефективне масштабування, що дозволяє збільшувати або зменшувати кількість активних ресурсів відповідно до потреб системи.

Ці проблеми загострюються в системах високої доступності, де мікросервіси зазвичай розгортаються на декількох вузлах з використанням єдиних базових образів. Це робить всю систему вразливою до атак, спрямованих на конкретні спільні вразливості.

Спільні вразливості становлять загрозу безпеці в архітектурах мікросервісів, особливо коли в багатьох екземплярах використовуються ідентичні образи контейнерів. Якщо в базовому образі контейнера існує вразливість, зловмисник може використати її у всіх екземплярах, що використовують цей образ. Це призведе до масової втрати функціональності сервісу. Така одноманітність контейнерів створює критичну точку відмови, яку необхідно усунути для підвищення стійкості системи.

Вирішення цих проблем полягає у збалансованому розподілі навантаження та розгортанні мікросервісів на різних вузлах та версіях. Забезпечуючи рівномірний розподіл екземплярів мікросервісів між вузлами, система може зменшити ризики, пов'язані з відмовою вузла. Завдяки використанню декількох версій образів контейнерів для кожного мікросервісу, система може запобігти впливу однієї вразливості на всі екземпляри. Такий підхід підвищує відмовостійкість, забезпечуючи різноманітність розгортання при збереженні балансу навантаження.

Методологія побудови моделі розгортання мікросервісів

Побудова моделі розгортання мікросервісів потребує комплексного підходу, який враховує як потреби в адаптивності, так і вимоги до відмовостійкості та оптимізації ресурсів. Методологія охоплює наступні основні етапи:

1. Аналіз вимог до системи:

- вимоги до відмовостійкості та високої доступності: визначається рівень відмовостійкості, якого необхідно досягти, і необхідні методи для забезпечення безперервної роботи всіх мікросервісів, навіть при відмові окремих вузлів;
- оцінка навантаження та потреб у масштабуванні: прогнозуються пікові навантаження на систему, на основі яких визначається, як система має масштабуватися горизонтально (через додавання нових вузлів) або вертикально (через збільшення ресурсів існуючих вузлів).
- вимоги до безпеки та захисту від вразливостей: ідентифікуються загальні вразливості компонентів і визначаються стратегії для їх ізоляції та зниження ризиків.

2. Вибір технологій та інструментів:

- контейнеризація: вибір платформи (наприклад, *Docker*), що дозволяє ізолювати сервіси, спрощуючи управління, оновлення та масштабування;
- система оркестрації: впровадження оркестратора (наприклад, *Kubernetes*), для автоматизації процесів управління контейнерами;
- управління конфігураціями: використання інструментів для зберігання конфігурацій і таємниць (наприклад, *HashiCorp Vault, Kubernetes ConfigMaps*), що забезпечують безпечний обмін конфіденційними даними між мікросервісами.

3. Розробка архітектури моделі розгортання:

- ізоляція мікросервісів: кожен мікросервіс розгортається як окремий контейнер, що дозволяє підтримувати ізоляцію та зменшує ризик передачі вразливостей між компонентами;
- створення кластерної архітектури: налаштування кластера, який складається з кількох вузлів для підвищення відмовостійкості;
- варіація навантаження: використання коефіцієнту варіації навантаження для розподілу запитів між сервісами, що дозволяє рівномірно розподіляти трафік, уникати перевантаження окремих вузлів і забезпечувати стійкість до пікових навантажень.

4. Забезпечення безпеки та управління вразливостями:

- сегментація мережі: впровадження політик мережевої ізоляції для обмеження доступу між мікросервісами, що зменшує ризики поширення атак всередині системи;
- оновлення образів контейнерів: регулярне оновлення базових образів контейнерів, що дозволяє усувати вразливості, які можуть вплинути на всі розгорнуті мікросервіси.
- використання служб для виявлення вразливостей: підключення до систем сканування вразливостей, таких як *Snyk* або *Trivy*, для автоматичного виявлення та виправлення вразливих місць у контейнерах і залежностях.

5. Налаштування моніторингу та логування:

- моніторинг продуктивності: впровадження інструментів для моніторингу стану сервісів і ресурсів (наприклад, *Prometheus* або *Grafana*). Це дозволяє своєчасно реагувати на проблеми з продуктивністю і відхилення від нормального стану.
- централізоване логування: налаштування логування для збору та аналізу журналів подій усіх мікросервісів у єдиному місці (наприклад, через *ELK Stack* або *Loki*). Це забезпечує простий доступ до логів, дозволяючи швидко виявляти проблеми.

Щоб оптимізувати розгортання мікросервісів і підвищити стійкість системи, спочатку вводимо два ключові показники ефективності: варіація навантаження і різноманітність. Ці показники слугують основою для максимізації відмовостійкості системи в умовах обмежень доступних обчислювальних ресурсів, зокрема процесора та пам'яті, а також підтвердили свою ефективність в дослідженні [9].

А. Коефіцієнт варіації навантаження

Для кожного мікросервісу, позначеного як MS_x, споживання ресурсів його *i*-м екземпляром на вузлі залежить від конкретної версії образу, яку він використовує. Обчислювальне навантаження на вузол можна кількісно оцінити за ресурсами процесора та пам'яті, які споживають екземпляри мікросервісів, розгорнуті на цьому вузлі.

Для оцінки рівномірності розподілу навантаження на процесор (*CPU*) і пам'ять (*MEM*) між вузлами використовуємо коефіцієнт варіації навантаження (*Load Variation Coefficient, LVC*). Цей коефіцієнт розраховується окремо для кожного типу ресурсу і показує, наскільки рівномірно розподілене навантаження між вузлами.

$$LVC_{CPU} = \frac{\sigma_{CPU}}{u_{CPU}},\ LVC_{MEM} = \frac{\sigma_{MEM}}{u_{MEM}},$$

де, $\overline{u_{CPU}}$ – середнє значення навантаження на процесор для всіх вузлів, σ_{CPU} –

середньоквадратичне відхилення навантаження на процесор, $\overline{u_{MEM}}$ – середнє значення навантаження на пам'ять, σ_{MEM} – середньоквадратичне відхилення навантаження на пам'ять.

Показники $\overline{u_{CPU}}$ та σ_{CPU} обраховуються так:

$$\overline{u_{CPU}} = \frac{1}{N}\sum_{j=1}^{N} u_{CPU,j}$$

$$\sigma_{CPU} = \sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(u_{CPU,j} - \overline{u_{CPU}}\right)^2},$$

де, $u_{CPU,j}$ – використання процесора на j -му вузлі, N – кількість вузлів.

Аналогічно обраховуються показники $\overline{u_{MEM}}$ та σ_{MEM}.

$$\overline{u_{MEM}} = \frac{1}{N}\sum_{j=1}^{N} u_{MEM,j},$$

$$\sigma_{MEM} = \sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(u_{MEM,j} - \overline{u_{MEM}}\right)^2},$$

де, $u_{MEM,j}$ – використання процесора на j -му вузлі, N – загальна кількість вузлів у системі.

Нарешті, кількісно оцінюємо загальний коефіцієнт навантаження:

$$LVC = \frac{LVC_{CPU} + LVC_{MEM}}{2}.$$

Цей коефіцієнт показує середню варіативність між ресурсами і дозволяє оцінити загальну рівномірність навантаження.

Б. Індикатор різноманіття

Різноманітність, як з точки зору розгортання на вузлах, як і з точки зору версій використовуваних образів, має вагоме значення для пом'якшення впливу спільних вразливостей. Різноманітність мікросервісу MS_x можна виміряти за допомогою різноманітності вузлів та різноманітності версій.

Різноманітність вузлів для мікросервісу обчислюється на основі розподілу його екземплярів між вузлами розгортання. Нехай p_j^{α} представляє частку екземплярів MS_x, розгорнутих на вузлі j -му, α – параметр ентропії. Тоді різноманітність вузлів виражається за допомогою ентропії Реньї як:

$$D_1 = \frac{1}{1-\alpha}\ln\left(\sum_{j=1}^{N} p_j^{\alpha}\right).$$

Аналогічно, різноманітність версій відображає, наскільки рівномірно розподілені екземпляри між різними версіями зображень. Нехай q_v^{α} позначає частку екземплярів, що використовують версію v зображення, а різноманітність версій задано через :

$$D_2 = \frac{1}{1-\alpha}\ln\left(\sum_{v=1}^{V} p_v^{\alpha}\right).$$

Загальний показник різноманітності для системи є середнім значенням між різноманітністю вузлів та різноманітністю версій:

$$D = \frac{D_1 + D_2}{2}.$$

В. Оптимізація

Для забезпечення відмовостійкості комбінуємо показники варіації навантаження та різноманітності в єдину метрику відмовостійкості E:

$$E = \gamma \cdot (1 - LVC) + (1-\gamma)\cdot D,$$

де, γ – ваговий коефіцієнт, який врівноважує важливість варіації навантаження та різноманітності. Мета полягає в тому, щоб максимізувати цю метрику стійкості, з урахуванням обмежень на доступні ресурси та різноманітність розгортання.

Г. Обмеження

Для забезпечення ефективного та надійного розгортання мікросервісів у системі введемо деякі ключові обмеження. Ці обмеження допомагають оптимізувати розподіл ресурсів, уникнути перевантаження вузлів та підвищити загальну стійкість системи. Щоб кожен екземпляр мікросервісу був розгорнутий лише на одному вузлі, введемо обмеження

$$\sum_{j=1}^{N} x_{i,j} = 1, \ \forall i.$$

Це обмеження запобігає дублюванню екземплярів на кількох вузлах і

сприяє рівномірному розподілу навантаження.

Для зменшення складності управління кожен екземпляр мікросервісу може використовувати лише одну версію контейнерного образу:

$$\sum_{v=1}^{V} y_{i,j} = 1, \ \forall i,$$

де, V – кількість доступних версій образів.

Це обмеження спрощує підтримку і оновлення образів, знижуючи ризик виникнення конфліктів між версіями.

Для кожного j -го вузла сумарне використання процесора та пам'яті для всіх розгорнутих екземплярів повинно задовольняти обмеженням:

$$\sum_{i=1}^{M} x_{i,j} \cdot u_{CPU,i} \le CPU_j, \ \forall j,$$
$$\sum_{i=1}^{M} x_{i,j} \cdot u_{MEM,i} \le MEM_j, \ \forall j,$$

де, $u_{CPU,i}$ – споживання процесора екземпляром i, CPU_j – максимальний доступний процесорний ресурс на вузлі j, $u_{MEM,i}$ – споживання пам'яті екземпляром i, MEM_j – максимальний доступний ресурс пам'яті на вузлі j, M – загальна кількість екземплярів мікросервісів у системі.

Ці обмеження гарантують, що розгортання є одночасно ресурсоефективним та стійким до відмов вузлів та спільних вразливостей.

Опис використаного алгоритму

Проблема оптимізації розгортання мікросервісів за своєю суттю є складною задачею через необхідність досягнення балансування навантаження та різноманітності. Це робить її *NP*-складною задачею, де пошук точного рішення вимагає значних обчислювальних витрат. Для її вирішення часто використовують евристичні алгоритми, такі як генетичні алгоритми та оптимізація рою частинок. Однак ці методи, як правило, мають високу часову складність і можуть займати значний час для збіжності.

В дослідженні використано евристичний алгоритм *LB-Diversity* [9], який поєднує принципи балансування навантаження з розгортанням на основі різноманітності. Мета алгоритму – децентралізувати розгортання мікросервісів, гарантуючи, що як екземпляри сервісів, так і їхні версії рівномірно розподілені між вузлами.

Алгоритм *LB-Diversity* використовує два ключові принципи:

1. Різноманітність вузлів. Екземпляри мікросервісів розподіляються між кількома вузлами, щоб уникнути концентрації на одному вузлі.
2. Різноманітність версій. Різні версії мікросервісів розгортаються для зменшення ризиків спільних вразливостей.

Алгоритм починається з сортування версій кожного мікросервісу на основі вимог до ресурсів. На вхід подається інформація про мікросервіс, вузол, версію сервісу та кількість розгортань. Менш ресурсоємні версії є пріоритетними для розгортання. Для кожного екземпляру мікросервісу алгоритм обирає вузол з мінімальним навантаженням і достатніми ресурсами для розгортання. На виході отримуємо остаточний план розгортання вузлів та результати розгортання мікросервісів.

Часова складність алгоритму в першу чергу визначається кількістю мікросервісів, їх версіями та доступними вузлами. Загальна часова складність може бути виражена як:

$$O=O(s[O(m_x \log m_x) + \\ +n_{deploy}(n_N O(n_N \log n_N))]),$$

де s – кількість типів мікросервісів; m_x – кількість версій для кожного мікросервісу; n_{deploy} – кількість розгортань; n_N – кількість вузлів.

Ця формула відображає складність сортування версій і вибору оптимального вузла розгортання на основі балансування навантаження і різноманітності.

Тестування

Для чистоти експерименту і для порівняння результатів з роботою [9] оцінка ефективності алгоритму порівнювалася з класичними алгоритмами, такими як *LB-Greedy*, *Round-Robin* і *Random* стратегії. Метрики оцінки, що використовувалися

для порівняння алгоритмів, включають коефіцієнти варіації навантаження, різноманіття, стійкості, варіації навантаження та продуктивності з метою перевірки адаптивності та ефективності алгоритму *LB-Diversity* в реальних сценаріях розгортання.

Система була реалізована з використанням технології *Software Defined Networking (SDN*) для управління та розгортання екземплярів мікросервісів. Контролер *SDN*, що виступає в ролі центрального вузла управління, відповідав за розподіл стратегій розгортання між серверними вузлами.

Для побудови мікросервісів було використано технологію контейнерів *Docker*, з різними базовими образами (*Ubuntu* (індекс 1), *Debian* (індекс 2)), що забезпечує різноманітність для розгортання функціональних компонентів. Кожен тип сервісу (*PHP*, база даних *MySQL*, система кешування *Redis*) мав дві різні реалізації на основі образів контейнерів.

Експерименти проводилися на сервері *Dell EMC R540* (2 процесори *Xeon Gold* 5220 та 256 ГБ оперативної пам'яті, 2 жорстких диски по 600 ГБ, під управлінням операційної системи *Ubuntu Server*).

На рис. 1 показано порівняння ефективності різних алгоритмів балансування навантаження для мікросервісів (*LB-Diversity*, *LB-Greedy*, *Random* і *Round-Robin*) в залежності від кількості розгортань мікросервісів. Алгоритм *LB-Diversity* демонструє найбільшу мінливість у показниках навантаження, що свідчить про її адаптивність у порівнянні з іншими методами. *LB-Diversity* має спади та підйоми, що може свідчити про її чутливість до змін у навантаженні та її спроби ефективно адаптуватися до різноманіття запитів.

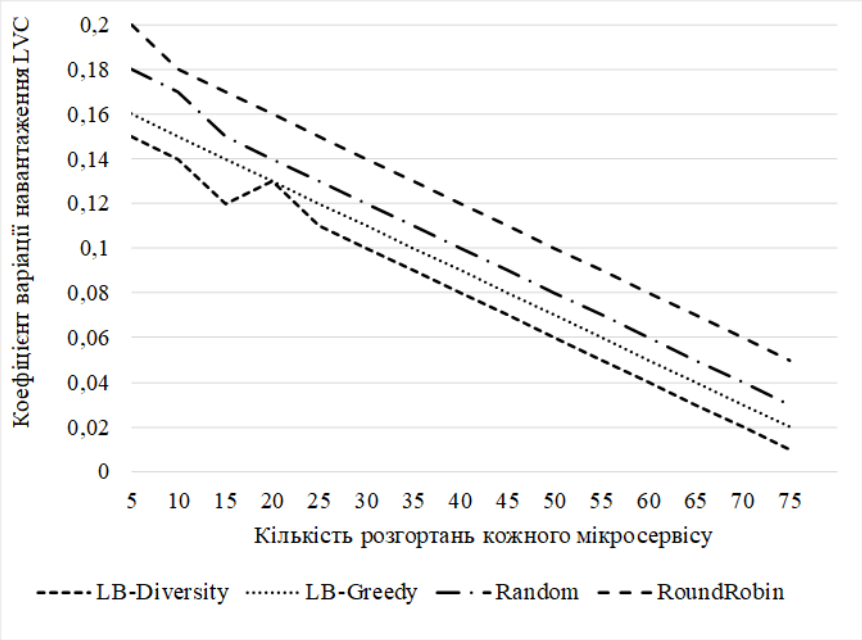

Рис. 1. Оцінка балансування навантаження в залежності від кількості розгортань

Графіки на рис. 2 показують порівняння показників різноманіття. Алгоритм *LB-Diversity* демонструє найвищий індикатор різноманіття, що стабілізується на рівні близько 3,2. Це свідчить про високу здатність *LB-Diversity* забезпечувати різноманітне розподілення запитів серед мікросервісів, що робить його ефективним для систем, де важливо підтримувати різноманітність і уникати надмірного навантаження на окремі вузли.

Алгоритм *LB-Diversity* також демонструє найвищий рівень стійкості (рис. 3), стабілізуючись на рівні близько 3,14. Це означає, що *LB-Diversity* є найбільш стійким у своєму розподілі навантаження, підтримуючи високий рівень стабільності навіть при збільшенні кількості розгортань.

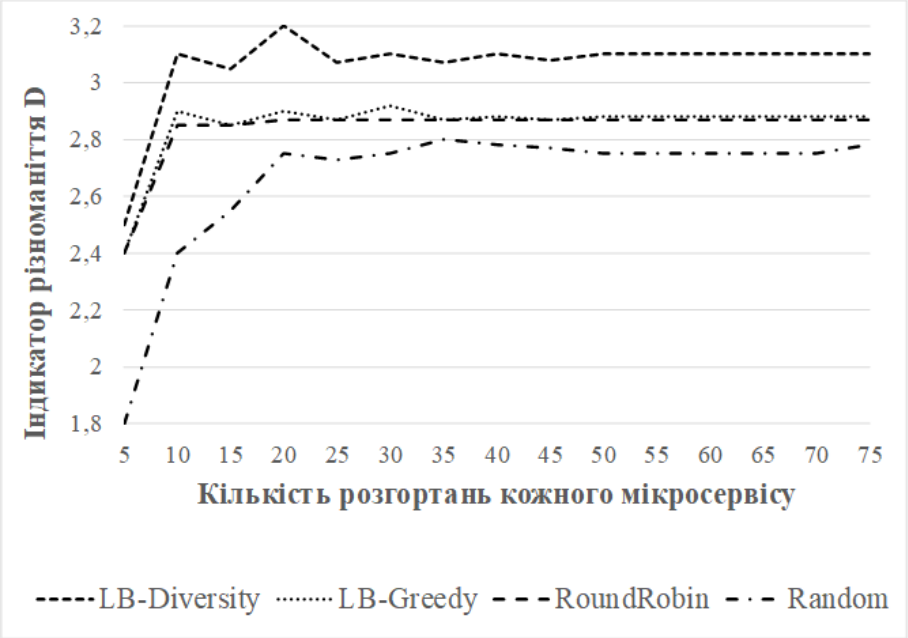

Рис. 2. Оцінка різноманіття в залежності від кількості розгортань

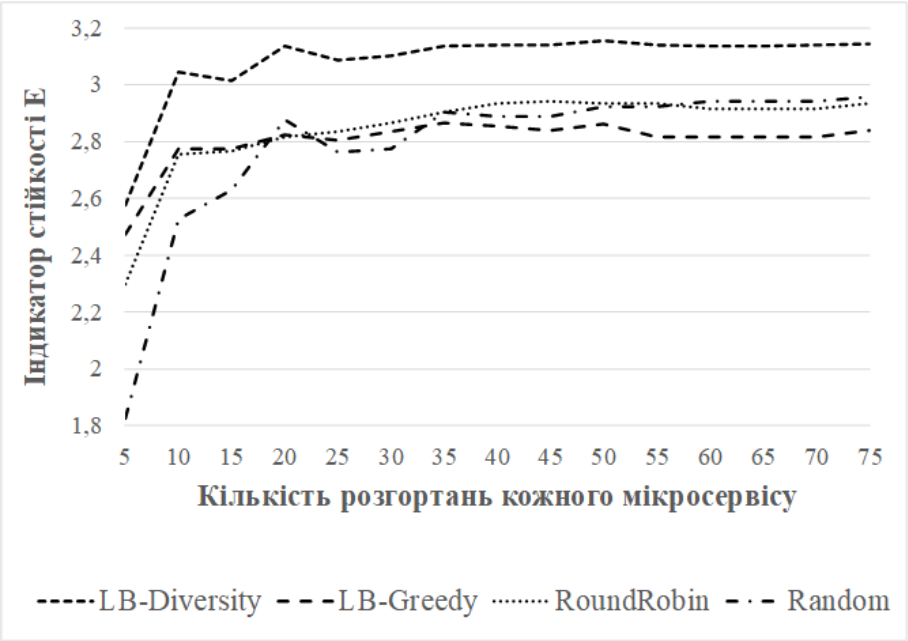

Рис. 3. Оцінка стійкості в залежності від кількості розгортань

При оцінці коефіцієнту варіації навантаження (рис. 4) алгоритм *LB-Diversity* демонструє відносно стабільне значення, яке коливається в межах від 0,15 до 0,3 залежно від кількості версій, хоча і відмічається стрибок на кількості версій 5, що свідчить про здатність підтримувати баланс навантаження навіть із збільшенням кількості версій мікросервісів, що є важливим для забезпечення рівномірного розподілу ресурсів.

За оцінкою рівня різноманіття серед усіх представлених стратегій (рис. 5) *LB-Diversity* має найвище значення. Із зростанням кількості версій для мікросервісів індикатор різноманіття для *LB-Diversity* стабільно збільшується, досягаючи значень, близьких до 2,9, що свідчить про високу ефективність алгоритму у підтриманні рівномірного та різноманітного розподілу навантаження.

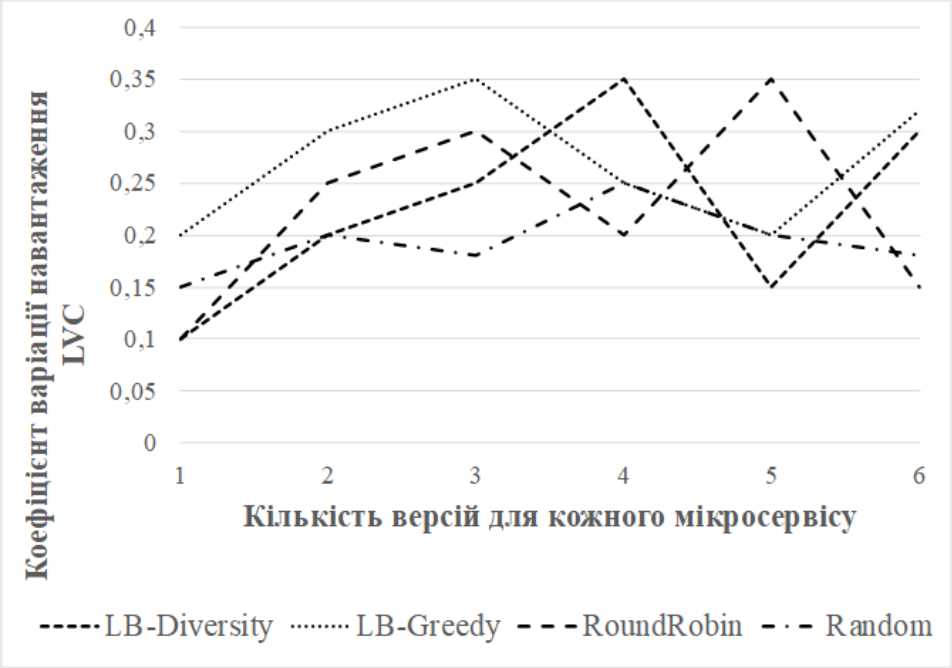

Рис. 4. Оцінка коефіцієнту варіації навантаження в залежності від кількості версій

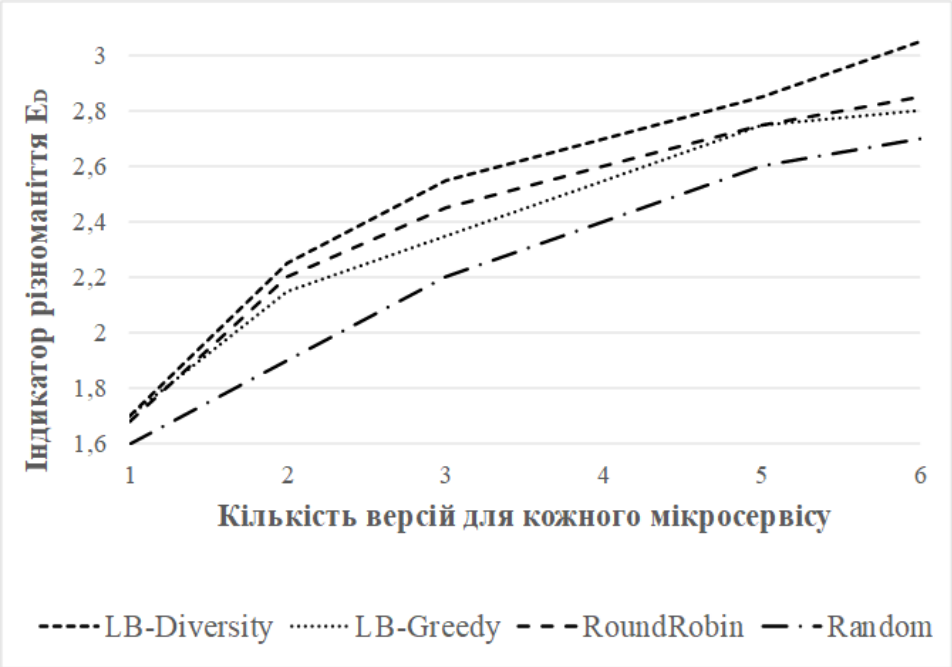

Рис. 5. Оцінка різноманіття в залежності від кількості версій мікросервісів

При тестуванні стійкості із збільшенням кількості версій (рис. 6) алгоритм *LB-Diversity* має найвищі показники (значення індикатора стійкості для нього поступово зростало і досягало найвищих показників на рівні близько 2,8), що свідчить про високу надійність у розподілі навантаження.

При тестуванні стійкості до втрат продуктивності в залежності від кількості недоступних вузлів (рис. 7) та в залежності від кількості базових образів (рис. 8) алгоритм *LB-Diversity* показав найкращі показники, що робить його оптимальним вибором для систем, де важливо підтримувати стабільну роботу при відмові вузлів, та є найбільш ефективною стратегією для підтримання продуктивності в умовах зростання кількості вразливих образів, тоді як *Random* демонструє найгірші результати.

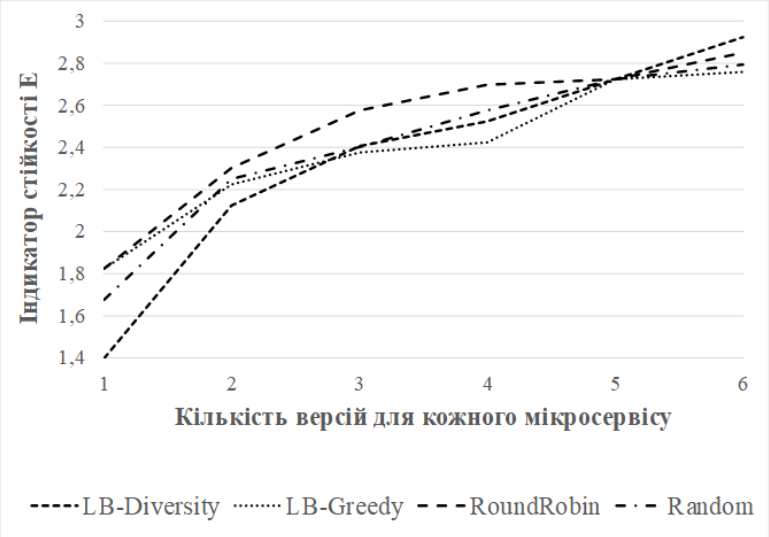

Рис. 6. Оцінка стійкості в залежності від кількості версій мікросервісів

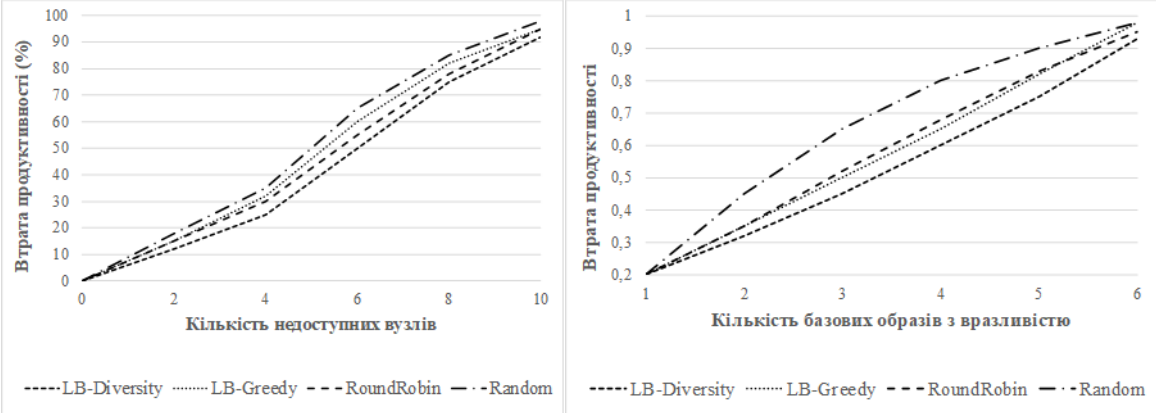

Рис. 7. Оцінка втрат продуктивності: а) в залежності від кількості недоступних вузлів, б) в залежності від кількості базових образів з вразливістю

Результати демонструють, що алгоритм *LB-Diversity* показав найкращі результати, порівняно з іншими традиційними стратегіями розгортання, як з точки зору балансування навантаження, так і з точки зору різноманіття. Застосовуючи децентралізоване розгортання як на вузлах, так і на версіях, *LB-Diversity* гарантує, що система залишається стійкою до відмов вузлів і спільних вразливостей, що призводить до кращої загальної продуктивності в різних сценаріях розгортання.

Висновки

Представлена модель розгортання мікросервісів, спрямована на оптимізацію відмовостійкості системи та оптимальне використання ресурсів. За допомогою спостереження за діючими системами було продемонстровано, що використання алгоритму *LB-Diversity* дозволило ефективно розподіляти навантаження між вузлами та зменшувати ризики, пов'язані з використанням ідентичних образів контейнерів. Здатність моделі розподіляти екземпляри між декількома версіями значно підвищує безпеку та надійність системи.

Алгоритм *LB-Diversity* продемонстрував значну перевагу в усіх ключових аспектах балансування навантаження в мікросервісних системах. За показниками продуктивності та стійкості до відмов, *LB-Diversity* перевершує інші стратегії, зокрема *LB-Greedy, Round-Robin* і *Random*. Згідно з результатами, *LB-Diversity* забезпечує на 15-20% меншу втрату продуктивності при відмовах вузлів, що критично важливо для систем із високими вимогами до доступності.

У ситуаціях, коли кількість недоступних вузлів зростала, *LB-Diversity* зберігав продуктивність із втратами, що на 10-12% менші, ніж у *LB-Greedy*, і на 25-30% нижчі порівняно з *Random*. Це свідчить про високу стійкість алгоритму до непередбачуваних збоїв, що дозволяє йому зберігати стабільну роботу навіть при значному зменшенні доступних ресурсів.

При наявності вразливих базових образів *LB-Diversity* також показав найкращі результати, забезпечуючи втрати продуктивності, які на 12-15% менші, ніж у *Round-Robin*, і на 20-25% менші, ніж у *Random*. Завдяки цьому *LB-Diversity* менш уразливий до загроз безпеки і краще пристосовується до роботи в умовах підвищеної кількості вразливих компонентів.

У середовищах із різними версіями мікросервісів *LB-Diversity* підтримує більш рівномірний розподіл навантаження, що дозволяє зберігати продуктивність на рівні, вищому на 10-15% порівняно з *LB-Greedy* і на 20-30% вищому порівняно з *Random*. Це робить його оптимальним вибором для систем, де швидка адаптація до змін є необхідною умовою.

Алгоритм *LB-Diversity* забезпечує стабільну і високу продуктивність, демонструючи на 15-30% кращі показники в порівнянні з іншими алгоритмами.

В майбутніх дослідженнях планується розглянути можливість інтеграції системи аналітики і машинного навчання в алгоритм *LB-Diversity* для подальшого підвищення її адаптивності. Такі вдосконалення можуть призвести до ще більшого скорочення часу виконання задач і більш ефективного розподілу ресурсів.

Література

1. Bravetti M. et al. Optimal and automated deployment for microservices. *Lecture Notes in Computer Science. Vol. 11424. Fundamental Approaches to Software Engineering. 22nd International Conference, FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings* / ed. by R. Hähnle, W. van der Aalst. Berlin, 2019. P. 351–368. DOI: 10.1007/978-3-030-16722-6_21.

2. Artamonov Y., Golovach I., Zymovchenko V. Use analysis of microserves in e-learning system with multi-variant access to educational materials. *Technology Audit and Production Reserves*. 2021. Vol. 4(2(60)). P. 45–50. DOI: 10.15587/2706-5448.2021.237760.

3. Fu Y. et al. Performance evaluation of resource management schemes for cloud-native platforms with computing containers. *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)* : proceedings, Austin, TX, USA, 11–13 November 2023 / IEEE. 2023. P. 414–415. DOI: 10.1109/IPCCC55026.2022.9894300.

4. Mustyala A. Dynamic resource allocation in Kubernetes: Optimizing cost and performance. *EPH – International Journal of Science and Engineering*. 2021. Vol. 7(3). P. 59–71. DOI: 10.53555/ephijse.v7i3.237.

5. González S. Modular software design in distributed systems: Strategic approaches for building scalable, maintainable, and fault-tolerant architectures in modern microservice environments. *Eigenpub Review of Science and Technology*. 2023. Vol. 7(1). P. 373–400. DOI: 10.1007/s10916-020-1195-x.

6. Mugeraya S., Devadkar K. Dynamic task scheduling and resource allocation for microservices in cloud. *Journal of Physics: Conference Series.* 2022. 2325. 012052. DOI: 10.1088/1742-6596/2325/1/012052.

7. Sebastião F. P. The role of a microservice architecture on cybersecurity and operational resilience in critical systems: master´s thesis. Porto, 2023. 190 p.

8. Mejía P. Best practices for microservice framework design. *Advances in Intelligent Information Systems*. 2022. Vol. 7(1). URL: https://questsquare.org/index.php/JOURNALAIIS/article/view/70.

9. Hang, Y. et al. A Microservice Resilience Deployment Mechanism Based on Diversity. *Security and Communication Networks*. 2022. 7146716. DOI: 10.1155/2022/7146716.

Артамонов Є.Б., Коцюр А.Б., Крант Д.В., Радченко К.М.

ПІДХІД ДО ОПТИМІЗАЦІЇ МОДЕЛІ РОЗГОРТАННЯ МІКРОСЕРВІСІВ В СИЛЬНОНАВАНТАЖЕНОМУ СЕРЕДОВИЩІ

У статті пропонується використання алгоритму LB-Diversity для розгортання мікросервісів з метою підвищення стійкості системи та оптимізації ресурсів у хмарних архітектурах. Традиційні мікросервісні системи часто стикаються з проблемами відмов вузлів та вразливостями безпеки через однорідне розгортання ідентичних екземплярів сервісів. Запропонований підхід вирішує ці проблеми, поєднуючи балансування навантаження з різноманітністю розгортання, коли різні версії мікросервісів розподіляються між кількома вузлами. Ця стратегія зменшує ризики, пов'язані з єдиною точкою відмови, і знижує вплив атак, спрямованих на спільні вразливості в контейнерних середовищах.

Для аналізу експериментів запропоновано власні коефіцієнти оцінки балансування навантаження та відмовостійкості системи, які підтвердили результати попередніх досліджень щодо вибору алгоритму LB-Diversity порівняно з традиційними методами розгортання. Підхід різноманіття розгортання ефективно обмежує шкоду від потенційних порушень безпеки, гарантуючи, що лише незначна частина системи зазнає впливу. Представлений підхід пропонує практичне рішення для підприємств, які потребують надійних мікросервісних архітектур з високим рівнем доступності та безпеки в динамічних середовищах з обмеженими ресурсами.

***Ключові слова**: архітектура мікросервісів; різноманітніття розгортання; балансування навантаження; стійкість системи; оптимізація ресурсів.*

Artamonov Y.B., Kotsiur A.B., Krant D.V., Radchenko K.M.

AN APPROACH TO OPTIMIZING THE MICROSERVICES DEPLOYMENT MODEL IN A HIGHLY LOADED ENVIRONMENT

The article proposes the use of a LB-Diversity algorithm for deploying microservices to increase system stability and optimize resources in cloud architectures. Traditional microservice systems often face node failure issues and security vulnerabilities due to homogeneous deployment of identical service instances. The proposed approach addresses these issues by combining load balancing with deployment diversity, where different versions of microservices are distributed across multiple nodes. This strategy reduces the risks associated with a single point of failure and mitigates the impact of attacks targeting common vulnerabilities in containerized environments.

To analyze the experiments, our own coefficients for assessing load balancing and system fault tolerance were proposed, which confirmed the results of previous studies on the choice of the LB-Diversity algorithm compared to traditional deployment methods. The diversity of deployment approach effectively limits the damage from potential security breaches by ensuring that only a small portion of the system is affected. This model offers a practical solution for enterprises that require reliable microservice architectures with high availability and security in dynamic environments with limited resources.

***Keywords**: microservice architecture; deployment diversity; load balancing; system resilience; resource optimization.*