DOI: https://doi.org/10.25140/2411-5363-2025-4(42)-208-217
УДК 004

Павло Володимирович Зінченко[1], Дмитро Борисович Мехед[2]

[1]аспірант кафедри інформаційних та комп'ютерних систем
Національний університет «Чернігівська політехніка» (Чернігів, Україна)
Email: zinchenkop@protonmail.com. **ORCID:** https://orcid.org/0009-0001-4573-6463. **ResearcherID:** MDS-7976-2025

[2]кандидат педагогічних наук, доцент кафедри кібербезпеки та математичного моделювання
Національний університет «Чернігівська політехніка» (Чернігів, Україна)
Email: d.mekhed@gmail.com. **ORCID:** https://orcid.org/0000-0003-3905-3620
ResearcherID: H-1751-2016. **SCOPUS Author ID:** 57193823626

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АВТОМАТИЧНОЇ ОПТИМІЗАЦІЇ МУЛЬТИРЕГІОНАЛЬНОЇ ІНФРАСТРУКТУРИ

У статті представлено інформаційну технологію динамічної оптимізації мультирегіональних хмарних розгортань, що ґрунтується на комплексному врахуванні експлуатаційних метрик затримки доступу та регіональної вартості обчислювальних ресурсів. Метою роботи є підвищення ефективності хмарних розгортань за рахунок автоматизованого вибору регіонів з урахуванням реального географічного розподілу користувачів і чинних тарифів хмарного провайдера. Запропоновано формальну модель, у якій ефективна затримка визначається як зважене мінімальне значення для кожної групи користувачів, а сумарна вартість — як агрегована вартість активних регіонів. На основі інтегрального критерію якості сформовано механізм прийняття рішень, який реалізовано у вигляді програмного агента. Агент здійснює автоматизований збір експлуатаційних метрик CDN і хмарних даних за ціною, оцінює поточний стан системи та ініціює зміну конфігурації за допомогою засобів інфраструктури як коду. Наведено кількісний приклад, що демонструє потенційне зменшення середньої ефективної затримки до 40 % та потенційну економію витрат до 2800 доларів США на рік за умови коректно підібраної мультирегіональної конфігурації. Отримані результати підтверджують доцільність використання динамічної оптимізації для підвищення продуктивності та економічної ефективності хмарних систем.

Ключові слова: *оптимізація; моніторинг; затримка; Cloudflare; AWS; інформаційна технологія; інфраструктура як код; Python; агент.*

Рис.: 1. Бібл.: 20.

Актуальність теми дослідження. Стрімке зростання глобальних вебсервісів призводить до того, що все більше застосунків одночасно обслуговують користувачів із кількох континентів. За таких умов ключовими стають два взаємопов'язані аспекти: затримка доступу до серверної інфраструктури та економічна доцільність вибору регіону її розгортання. Відомо, що різниця у вартості однакових інстансів хмарного провайдера в різних регіонах може становити десятки відсотків, а інколи й перевищувати двократне значення [1]. Водночас реальна затримка між користувачем і дата-центром здатна різко змінюватися залежно від географічного розподілу трафіку, часу доби та завантаженості мережевих маршрутів.

Сучасні CDN-платформи надають деталізовані експлуатаційні метрики, які відображають справжню географію користувачів. Хмарні провайдери, у свою чергу, відкривають API для отримання актуальних цін на інфраструктурні ресурси [2–4]. Попри це, більшість сервісів продовжують працювати з фіксованою регіональною конфігурацією, не використовуючи можливість адаптивної перебудови архітектури у відповідь на зміни в поведінці трафіку або вартості ресурсів.

У таких умовах актуальною стає розробка технології, здатної автоматично інтегрувати дані про затримку й вартість у єдиний механізм оптимізації мультирегіональних розгортань, що й становить предмет цього дослідження.

Постановка проблеми. Проблема полягає у відсутності практичного інструменту, який міг би динамічно визначати оптимальне розташування хмарної інфраструктури на основі реальних експлуатаційних метрик. У сучасних системах вибір регіону зазвичай здійснюється разово – на етапі проєктування – і рідко змінюється надалі. Такий статичний підхід призводить або до надмірних витрат, коли сервіс працює у дорогому регіоні без потреби, або до підвищеної затримки, коли інфраструктура розташована далеко від фактичних точок доступу.

Навіть коли адміністратори мають доступ до CDN-аналітики та до інструментів моніторингу вартості, вони переважно аналізують такі дані вручну. Крім того, відсутній формальний механізм, який би дозволяв інтегрувати метрики про затримку, регіональні тарифи та параметри міграції інфраструктури у цілісну модель прийняття рішень. У результаті регіональні зміни або не виконуються взагалі, або проводяться епізодично та без гарантій оптимальності.

Проблема ускладнюється тим, що зміна регіону – це не лише вибір місця, але й динамічний перехід, що має власну вартість і ризики. Отже, виникає потреба в технології, здатній не лише оцінити стан системи, а й автоматично адаптувати інфраструктуру на основі формальної моделі.

Аналіз останніх досліджень та публікацій. У науковій літературі широко обговорюються методи оптимізації розміщення сервісів у хмарних, fog-орієнтованих та edge-орієнтованих середовищах [5–10]. Числові моделі, що використовують багатокритеріальну оптимізацію, пропонують враховувати затримку, навантаження на вузли, енергоспоживання та інші параметри [5–9; 13; 14]. Однак такі дослідження здебільшого виконані у симуляційних умовах і не залучають реальних експлуатаційних метрик CDN або даних про актуальні ціни хмарних провайдерів. Отже, вони слабо застосовні до реальних систем, у яких інтенсивність трафіку й географія користувачів змінюються динамічно.

Практичні рекомендації хмарних провайдерів зосереджені переважно на статичному виборі регіону: мінімізація затримки, відповідність регуляціям, доступність сервісів та вартість [1; 11]. Проте жоден з цих підходів не пропонує механізмів, які дозволяють змінювати конфігурацію інфраструктури у процесі експлуатації. FinOps-підходи, хоча й приділяють значну увагу оптимізації вартості, фокусуються здебільшого на внутрішньорегіональних змінах (типи інстансів, резервування, Spot-моделі), а не на динамічному виборі регіону [12].

Що стосується CDN-аналітики, то хоча платформи на кшталт Cloudflare надають деталізовані метріки затримки, ці дані найчастіше використовуються для побудови дашбордів, а не у складі інтегрованої технології автоматичного прийняття рішень [3; 4]. Таким чином, у наявних дослідженнях бракує рішень, які б об’єднували метрики затримки та ціни у єдиний механізм оптимізації, здатний працювати у реальних експлуатаційних умовах.

Виділення недосліджених частин загальної проблеми. Попри активний розвиток методів оптимізації розміщення сервісів у хмарних та edge-орієнтованих інфраструктурах, важлива частина проблеми залишається нерозкритою — а саме, використання реальних експлуатаційних метрик для динамічного керування мультирегіональними розгортаннями [5–8; 13; 14]. Більшість існуючих підходів ґрунтуються або на аналітичних моделях у симуляційному середовищі, або на узагальнених рекомендаціях хмарних провайдерів щодо вибору регіону. Такі моделі не враховують фактичної географії трафіку, реальних показників затримки та актуальної вартості ресурсів у кожному регіоні — параметрів, які суттєво змінюються залежно від часу доби, континенту чи змін у поведінці користувачів.

Недостатньо дослідженою залишається й інтеграція CDN-аналітики, зокрема метрик Cloudflare, із механізмами автоматичного керування хмарною інфраструктурою [3,4]. Хоча CDN-провайдери надають високоточні вимірювання затримки та детальний розподіл запитів за географією, ці дані майже не застосовуються у процесі автоматичного вибору або корекції регіонів розгортання. У наявних роботах не розглядається можливість поєднання таких метрик у формальну модель, здатну оцінювати ефективну затримку для всієї системи загалом.

Окремою прогалиною є недостатня увага до економічних аспектів, пов’язаних із динамічним перемиканням між регіонами. Питання вартості міграції, міжрегіонального трафіку, збільшення обчислювального навантаження під час масштабування або згортання інфраструктури – у більшості досліджень або ігноруються, або враховуються дуже

приблизно [15–18]. Це створює разючу невідповідність між теоретичними моделями та реальними умовами роботи хмарних сервісів.

Не менш важливим є те, що в літературі фактично відсутні рішення, які пропонують завершений технологічний цикл: від збору даних до автоматизованої зміни конфігурації засобами інфраструктури як коду. Більшість публікацій зосереджуються на математичних моделях оптимізації або окремих рекомендаціях FinOps-підходів, але майже не торкаються питання реалізації програмного агента, здатного працювати в реальних умовах і адаптувати архітектуру системи відповідно до змін у трафіку та вартості ресурсів.

Таким чином, основним недослідженим напрямом залишається створення інтегрованої технології, яка поєднує метрики CDN, регіональні прайсинг-дані та механізми автоматичного розгортання в одну цілісну модель ухвалення рішень. Саме ця прогалина й визначає актуальність розроблення підходу, який буде представлений у подальших розділах статті.

Мета статті. Метою статті є розроблення та обґрунтування інформаційної технології динамічної оптимізації мультирегіональних хмарних розгортань, яка враховує одночасно метрики затримки, географічний профіль користувачів і реальні регіональні витрати на інфраструктуру. У межах цієї мети передбачається створити формальну математичну модель для обчислення ефективної затримки та сумарної вартості конфігурації, визначити інтегральну функцію якості, що поєднує обидва критерії, та запропонувати механізм вибору оптимального набору регіонів залежно від їхнього внеску в загальну продуктивність і фінансові показники.

Додатковою частиною мети є побудова архітектури програмного агента, який автоматично збирає експлуатаційні метрики з CDN та хмари, оцінює поточну конфігурацію за розробленою моделлю, генерує альтернативи й може ініціювати відповідні зміни за допомогою засобів інфраструктури як коду. Запропонований підхід має продемонструвати, що інтеграція метрик затримки та ціни дозволяє адаптивно керувати мультирегіональною інфраструктурою, покращуючи співвідношення швидкодії та витрат у порівнянні з традиційними статичними підходами.

Виклад основного матеріалу. Запропонована інформаційна технологія базується на ідеї замкненого контуру керування конфігурацією мультирегіональної інфраструктури. У цьому контурі агент:

• безперервно збирає метрики трафіку та затримки з рівня CDN (Cloudflare) і дані про вартість ресурсів у різних регіонах хмарного провайдера [2–4];

• на основі цих даних оцінює поточну конфігурацію розгортання та обирає найбільш доцільний варіант з огляду на компроміс між затримкою та вартістю;

• генерує альтернативні варіанти (наприклад, додати новий регіон, вимкнути частину інфраструктури);

• за потреби автоматично застосовує зміни до інфраструктури за допомогою засобів інфраструктури як коду (Pulumi тощо).

Модель роботи агента. На рівні збору даних агент підключається до Cloudflare API та отримує статистику за країнами, регіонами та дата-центрами: кількість запитів, частку трафіку, показники затримки (наприклад, медіану або 95-й перцентиль). Cloudflare Web Analytics дають такі дані з досить високою деталізацією в розрізі регіонів і періодів часу [3; 4].

Паралельно агент використовує AWS Price List API або зовнішні агрегатори цін (наприклад, cloudprice.net, aws-pricing.com) для отримання актуальної вартості інстансів вибраного типу в кожному регіоні [2; 19]. Для інстансу t4g.medium (2 vCPU, 4 GiB RAM) агреговані дані показують, що мінімальна ціна on-demand становить близько 0,0224 $/год, середня – приблизно 0,0388 $/год, максимальна – до 0,0536 $/год. У перерахунку на місяць (24 год × 30 днів) це дає від ≈16,3 $/міс до ≈39 $/міс, причому реальні сервіси для порівняння інстансів вказують для t4g.medium типові значення на рівні ≈16,35 $/міс у найдешевших регіонах і ≈28,2 $/міс як близьке до середнього значення [2; 19].

Отримані дані проєктуються на модель, де:

• кожен регіон AWS описується оцінкою вартості *C(r)* для заданого обсягу ресурсів (наприклад, один або *N* інстансів t4g.medium) та профілем затримки до основних груп користувачів;

• кожна група користувачів (країна або сукупність країн) характеризується часткою трафіку *pu* та виміряною або оціненою затримкою *L(r,u)* до кожного регіону *r*.

Для заданої множини активних регіонів *S* агент обчислює середню ефективну затримку як середнє по всіх групах користувачів:

$$Leff(S) = \sum_{u} pu \min_{r \in S} L(r,u),$$

а сумарну вартість конфігурації – як суму вартостей усіх активних регіонів:

$$C_{tot}(S) = \sum_{r \in S} C(r).$$

Таким чином, кожна конфігурація *S* описується парою $L_{eff}(S)$, $C_{tot}(S)$. Далі агент порівнює поточну конфігурацію з альтернативними: наприклад, добудувати другий регіон поблизу нового кластера користувачів; навпаки, відмовитися від регіону, який істотно збільшує витрати, але майже не покращує затримку; перенести основне навантаження в дешевший регіон за умови, що середня затримка залишиться в допустимих межах.

Щоб уникнути «миготіння» між регіонами, використовується гістерезис: конфігурація змінюється лише тоді, коли очікуване покращення (наприклад, зниження $L_{eff}(S)$ або $C_{tot}(S)$) перевищує наперед заданий поріг, а також коли виконується перевірка на окупність з урахуванням вартості міграції. Практичні матеріали з AWS наголошують, що міжрегіональний трафік і операції реплікації можуть суттєво збільшувати рахунок, тому ці витрати мають явно враховуватися агентом [1; 11].

На рівні виконання зміни здійснюються через Pulumi (або інші IaC-інструменти), які дозволяють програмно описати EC2-інстанси, VPC, балансувальники, а також інтеграцію з Cloudflare. Такий підхід узгоджується з сучасними DevOps- та FinOps-практиками, де інфраструктура описується як код і може модифікуватися автоматизовано на основі зовнішніх сигналів [12].

Архітектура агента. З огляду на те що, агент має три основні функції: збір статистики, аналіз та застосування змін, то доцільно розбити архітектуру агенту на три основні функціональні блоки. Перелік функціональних блоків для кожної із функцій:

• Collector – збір статистики та конвертація її у зручний формат для застосування розробленої моделі;

• Analazyer – використовуючи розроблену модель із врахуванням конфігурації сервісу знаходження оптимальної мультирегіональної конфігурації інфраструктури;

• Deployer – пропозиція змін на основі шаблону, розгортання запропонованих змін та оновлення поточного стану;

Крім функціональних блоків, потрібно визначити, яким чином будуть задаватися параметри агенту, адже агент повинен розуміти, яку статистику збирати, а також розуміти яку інфраструктуру розгортувати в додатковому регіоні. Тож, для роботи цих функціональних блоків на вході агент отримує 2 блоки налаштувань та 2 блоки опису IaC ресурсів:

• конфігурація агенту – як часто виконувати оптимізацію, налаштування посилань на API для збору статистики, налаштування логування та інші налаштування, що стосуються здебільшого роботи самого агенту, а не бізнес-логіки;

• конфігурація сервісу для моніторингу та оптимізації - як отримати статистику саме для потрібного сервісу, поріг допустимої затримки, мінімальне покращення для застосування;

• опис шаблону ресурсів для розгортання в межах регіону – опис ресурсів котрі потрібно розгортувати в межах нового регіону;

• опис поточного стану ресурсів до та після оптимізації – опис поточного стану ресурсів для визначення стану розгорнутих регіонів та ресурсів для них.

Першим функціональним блоком виступає Collector, який збирає статистику про стан сервісу – затримку та витрати на нього. Збір статистки виконується за допомогою API Cloudflare або AWS для AWS Cloudfront, посилання на які вказані в загальному конфігураційному файлі, де також визначено частота отримання статистики. Агент збирає статистичні дані тільки для заданого фільтра, який визначений у конфігурації для сервісу. Отримані дані статистки формуються у зручний формат для аналізу й передаються до блоку Analyzer.

Функціональний блок Analyzer на основі статистки та заданих параметрів в конфігурації сервісів (таких як поріг допустимої затримки та мінімальне покращення для застосування), виконує оптимізацію, тобто визначає доцільність змін у регіональному розміщенні: наприклад, додати регіон зі зростанням трафіку, прибрати регіон із низькою завантаженістю.

Після ухвалення оптимізаційних рішень блок Deployer формує зміни та застосовує їх. Він порівнює перевіряє розгорнуті регіони й у разі потреби виконує розгортання змін: додавання або видалення регіону. На виході оновлена інфраструктура фіксується в актуальному окремому IaC стані, забезпечуючи прозорий цикл змін і можливість подальшого контролю, а також перезапуск програмного агента без втрати стану сервісу.

Описану архітектуру можна побачити на рис. 1.

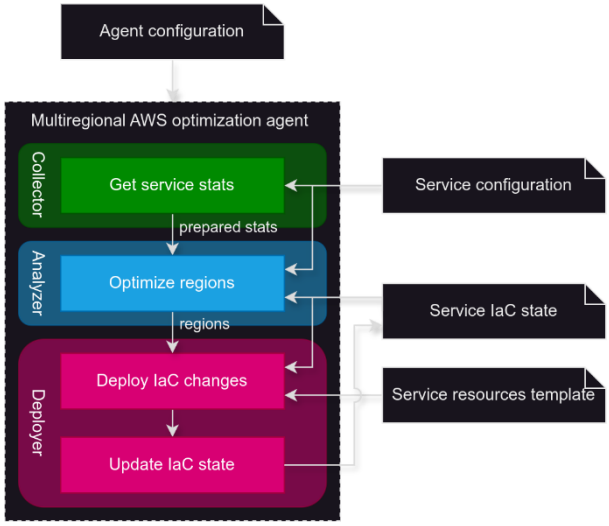

Рис. 1. Архітектура програмного агента

Реалізація програмного агента виконана мовою програмування Python. Для реалізації описаної архітектури програмного агента були написані такі функції:

• init_logger() - ініціалізація об'єктів для логування і подальше використання їх у всій програмі;

• init_settings() - ініціалізація налаштувань самого агента й подальше використання завантажених параметрів із оточення;

• init_configuration() - завантаження конфігурації сервісу;

• init_pulumi_state() - ініціалізація пустого стейту pulumi та імпортування вже існуючої попередньо експортованого стейту сервісу;

• get_instances_price() - отримання цін на інстанси для подальшого розрахунку ефективності розгорування регіонів;

• make_cloudflare_request() - формування та пейджинг прямих запитів до Cloudflare GraphQL API;

• make_cloudprice_request() - формування прямих запитів до CloudPrice API щодо цін на інстанси;

• get_raw_stats() - отримання непідготовленої статистики використання сервісу;

• prepare_stats(raw_stats: dict) - формування непідготовленої статистики у зручний для аналізу формат;

• get_service_usage() - отримати підготовлену статистику для сервісу;

• get_regions_from_state() - отримати поточний список регіонів для подальшої оптимізації;

• optimize_regions(stats: dict) - оптимізація регіонів за допомогою моделі: визначення списку актуальних регіонів;

• deploy_changes(regions: list) - розгорнути актуальний список регіонів, інші регіони видалити;

• update_resources_state() - оновити окремий спеціальний стейт для візуалізації для ops команд;

• main() - основний цикл роботи агенту.

Кількісний приклад оцінки ефективності. Щоб продемонструвати ефективність підходу, розглянемо спрощений, але чисельно конкретний сценарій. Нехай вебзастосунок обслуговує користувачів з Європи та Південної Азії. За даними Cloudflare аналітики, у середньому 70 % запитів надходить із країн ЄС і Великої Британії, а 30 % – з Індії та сусідніх країн [3; 4].

Для інфраструктури використовується інстанс типу t4g.medium. Згідно з агрегованими даними по регіонах, найдешевші регіони пропонують орієнтовно ≈16,35 $/міс за один t4g.medium, тоді як більш «типові» європейські регіони дають значення на рівні близько 28 $/міс [2; 19].

Для оцінки затримки скористаємося публічними таблицями RTT між регіонами та континентами, які показують, що [20]:

• затримка всередині Європи (користувачі в ЄС → регіон eu-central-1, Frankfurt) зазвичай знаходиться в діапазоні 30-60 мс;

• затримка між Західною Європою та Південною Азією (наприклад, до регіону ap-south-1, Mumbai) може сягати 170-220 мс;

• у зворотному напрямку (користувачі в Індії → регіон у Європі) спостерігаються подібні величини, тоді як до найближчого регіону в Азії затримка знижується до 50–70 мс.

Розглянемо три варіанти конфігурації.

Варіант А (один європейський регіон). Застосунок розгорнуто лише в регіоні eu-central-1 (Frankfurt), вартість одного t4g.medium — приблизно 28 $/міс. Для оцінки візьмемо такі усереднені значення:

• для європейських користувачів – 50 мс до Frankfurt;

• для користувачів з Південної Азії – 180 мс до Frankfurt.

Тоді середня ефективна затримка становитиме:

$$L_{eff}(A) = 0{,}7 \cdot 50 + 0{,}3 \cdot 180 \approx 35 + 54 = 89\ ms.$$

Сумарна вартість:

$$C_{tot}(A) \approx 28 \text{ dollars/month}.$$

Варіант B (один азійський регіон). Розгортаємо застосунок лише в регіоні ap-south-1 (Mumbai). За даними агрегаторів цін, для цього регіону t4g.medium знаходиться ближче до мінімальної межі, тобто ≈16,5 $/міс [2; 19]. Припустимо:

• для користувачів у Європі – ≈200 мс до Mumbai;

• для користувачів Південної Азії – 60 мс.

Тоді:

$$L_{eff}(B) = 0{,}7 \cdot 200 + 0{,}3 \cdot 60 = 140 + 18 = 158\ ms,\ C_{tot}(B) \approx 16{,}5 \text{ dollars/month}.$$

Отже, порівняно з варіантом A, вартість знижується майже на 40 %, але середня затримка зростає майже вдвічі. Для інтерактивних сервісів це, як правило, неприйнятно.

Варіант C (два регіони, Європа + Південна Азія). Розгортаємо по одному t4g.medium у Frankfurt і Mumbai, а Cloudflare Load Balancer розподіляє трафік за географічним принципом: європейські користувачі обслуговуються з європейського регіону, азійські — з ap-south-1 [3; 4]. Вартість:

$$C_{tot}(C) \approx 28 + 16{,}5 = 44{,}5 \text{ dollars/month}.$$

Для затримки:

• Європа → Frankfurt: 50 мс;

• Південна Азія → Mumbai: 60 мс.

Середня ефективна затримка:

$$L_{eff}(C) = 0{,}7 \cdot 50 + 0{,}3 \cdot 60 = 35 + 18 = 53\ ms.$$

Якщо порівняти варіанти:

• перехід від A до C зменшує середню затримку приблизно з 89 мс до 53 мс (близько – 40 %), але збільшує вартість у 1,6 раза (з 28 $/міс до 44,5 $/міс);

• перехід від B до C підвищує вартість (з 16,5 $/міс до 44,5 $/міс), але зменшує середню затримку зі 158 мс до 53 мс (понад утричі).

У реальній системі вибір між цими сценаріями залежатиме від допустимих рівнів затримки та бюджету. Запропонована технологія дозволяє формалізувати це як задачу мінімізації деякої інтегральної функції якості $F(S)$, наприклад:

$$F(S) = \alpha \cdot L_{eef}(S) + \beta \cdot C_{tot}(S),$$

де коефіцієнти α і β відображають пріоритети системи: чутливість до затримки і до вартості. Якщо, наприклад, сервіс належить до класу, де мінімізація затримки критична (онлайн-ігри, біржові системи), ваговий коефіцієнт при L_{eff} буде високим, і агент віддасть перевагу конфігурації C. Для менш чутливих до часу відгуку бізнес-застосунків із жорстким бюджетом може бути обрано варіант A або навіть B.

На горизонті року різниця у вартості стає відчутнішою. Один інстанс t4g.medium, запущений у середньодорогому регіоні (≈28 $/міс), обійдеться приблизно в 336 $/рік, тоді як у дешевшому регіоні (≈16,5 $/міс) — близько 198 $/рік. Для кластера з 20 інстансів потенційна економія може сягати ~2800 $/рік, якщо агент виявить сценарії, де перехід у дешевший регіон не призводить до критичного зростання затримки. Такі розрахунки добре узгоджуються з оцінками FinOps-практиків щодо впливу вибору регіону на річний бюджет [12].

Отже, навіть у простому прикладі видно, що поєднання метрик затримки та вартості дає змогу виявити конфігурації, які суттєво покращують один із показників за прийнятного зростання іншого. Запропонований агент формалізує цей процес, автоматизує збір і аналіз даних та, на відміну від ручного налаштування, здатний реагувати на зміни географії трафіка протягом експлуатації системи, а не лише на етапі початкового проєктування.

Висновки. У роботі запропоновано інформаційну технологію, яка дозволяє динамічно оптимізувати мультирегіональні хмарні розгортання на основі поєднання метрик затримки

та регіональної вартості ресурсів. На відміну від статичних архітектурних підходів, запропонована технологія інтегрує реальні експлуатаційні дані CDN, дані хмарних прайсинг-API та формальну модель оцінки конфігурацій, що включає середню ефективну затримку, загальну вартість та інтегральний критерій якості. Застосування такої моделі у складі програмного агента дає можливість не лише аналізувати поточний стан системи, але й ініціювати автоматизовану зміну регіонів розгортання через засоби інфраструктури як коду.

Кількісний приклад, наведений у статті, демонструє, що навіть у простих сценаріях поєднання двох регіонів може суттєво знизити середню затримку для глобальної аудиторії, тоді як перехід у дешевші регіони — забезпечити відчутну економію при збереженні прийнятного рівня продуктивності. Це підтверджує доцільність застосування розробленої технології в умовах змінної географії трафіку та високої варіативності регіональної вартості хмарних ресурсів.

Своєю чергою варто зазначити, що запропонований підхід є доволі декларативним на рівні описаних ресурсів для сервісу в шаблоні, хоча на рівні регіонів підхід дозволяє динамічно виконувати оптимізацію розгорнутої інфраструктури.

Окремим мінусом інформаційної технології в поточній реалізації є те, що її складно застосувати із сервісами котрі мають зберігати стан, наприклад, у кластерних базах даних.

Отримані результати свідчать, що перехід від статичного до адаптивного керування мультирегіональною інфраструктурою є ефективним шляхом підвищення якості обслуговування та зменшення експлуатаційних витрат. Подальші дослідження можуть бути спрямовані на розширення моделі прогнозними механізмами та застосуванням методів машинного навчання для передбачення трафіку й автоматичного формування оптимальних конфігурацій, незважаючи на сервіси зі збереженим станом чи ні.

Заява про використання генеративного ШІ та технологій на основі ШІ в процесі написання текстів.

Під час написання цієї статті автори використовували ChatGPT для упорядкування думок стосовно постановки проблеми, пошуку та аналізу схожих наукових робіт і з метою усунення стилістичних та граматичних помилок. Після використання цього інструменту автори переглянули та відредагували зміст за потреби і взяли на себе повну відповідальність за зміст публікації.

Список використаних джерел

1. Amazon Web Services. (2022). *COST07-BP02 Choose regions based on cost*. AWS Well-Architected Framework Documentation. https://docs.aws.amazon.com/wellarchitected/latest/cost-optimization-pillar/cost_pricing_model_region_cost.html.

2. Amazon Web Services. (2024). *AWS Price List Service and Price List API documentation*. https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/bulk-api-reading-price-list-files.html.

3. Cloudflare. (2024). *Load Balancing and Load Balancing Analytics documentation*. https://developers.cloudflare.com/load-balancing/reference/load-balancing-analytics/.

4. Cloudflare. (2024). *Web Analytics documentation*. https://developers.cloudflare.com/web-analytics/

5. Aslanpour, M. S., Gill, S. S., & Toosi, A. N. (2020). Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. *Internet of Things, 12*, 100273.

6. Taleb, I., Guillaume, J.-L., & Duthil, B. (2025). A survey on services placement algorithms in integrated cloud–fog/edge computing. *ACM Computing Surveys, 57*(11), Article 268.

7. Apat, H. K., Sahoo, B., Maiti, P., Barik, R. K., & Saikia, M. J. (2025). Fog service placement optimization: A survey of state-of-the-art strategies and techniques. *Computers, 14*(3), 99.

8. Supraja, T., Chawla, P., & Gill, S. S. (2025). AI-driven service placement in fog and edge computing environments: A systematic review, taxonomy and future directions. *Cluster Computing, 28*, 1070.

9. Velasquez, K., Perez Abreu, D., Curado, M., & Monteiro, E. (2021). Service placement for latency reduction in the fog using application profiles. *IEEE Access, 9*, 80821–80834.

10. Guerrero, C., Lera, I., & Juiz, C. (2018). A lightweight decentralized service placement policy for performance optimization in fog computing. *Journal of Ambient Intelligence and Humanized Computing, 9*(5), 1467–1482.

11. Guerrero, C., Lera, I., & Juiz, C. (2019). Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures. *Future Generation Computer Systems, 97*, 131–144.

12. Ghobaei-Arani, M., & Shahidinejad, A. (2022). A cost-efficient IoT service placement approach using whale optimization algorithm in fog computing environment. *Expert Systems with Applications, 200*, 117012.

13. Chikhaoui, A., Lemarchand, L., Boukhalfa, K., & Boukhobza, J. (2021). Multi-objective optimization of data placement in a Storage-as-a-Service federated cloud. *ACM Transactions on Storage, 17*(3), Article 22.

14. Liu, X., Fan, L., Wang, L., et al. (2016). Multiobjective reliable cloud storage with its particle swarm optimization algorithm. *Mathematical Problems in Engineering, 2016*, 9529526.

15. Adhikari, M., Mukherjee, M., & Srirama, S. N. (2020). DPTO: A deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing. *IEEE Internet of Things Journal, 7*(7), 5773–5782.

16. Boubaker, N. E. H., Zarour, K., Guermouche, N., & Benmerzoug, D. (2022). Fog and edge service migration approaches based on machine learning techniques. In *Proceedings of the Tunisian–Algerian Conference on Applied Computing (TACC)* (pp. 33–44). CEUR Workshop Proceedings, 3333.

17. Amazon Web Services. (2021). *What to consider when selecting a region for your workloads*. AWS Architecture Blog. https://aws.amazon.com/blogs/architecture/what-to-consider-when-selecting-a-region-for-your-workloads/.

18. FinOps Foundation. (2023). *Managing cloud cost anomalies and related FinOps capabilities*. https://www.finops.org/wg/managing-cloud-cost-anomalies/.

19. Cloudprice.net. (2025). *t4g.medium specs and pricing | AWS*. https://cloudprice.net/.

20. Bichard, L. (2021). How to find which AWS region is closest to you? *DEV Community*. https://dev.to/loujaybee/how-to-find-which-aws-region-is-closest-to-you-3k38.

References

1. Amazon Web Services. (2022). *COST07-BP02 Choose regions based on cost*. AWS Well-Architected Framework Documentation. https://docs.aws.amazon.com/wellarchitected/latest/cost-optimization-pillar/cost_pricing_model_region_cost.html.

2. Amazon Web Services. (2024). *AWS Price List Service and Price List API documentation*. https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/bulk-api-reading-price-list-files.html.

3. Cloudflare. (2024). *Load Balancing and Load Balancing Analytics documentation*. https://developers.cloudflare.com/load-balancing/reference/load-balancing-analytics/.

4. Cloudflare. (2024). *Web Analytics documentation*. https://developers.cloudflare.com/web-analytics/

5. Aslanpour, M. S., Gill, S. S., & Toosi, A. N. (2020). Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. *Internet of Things, 12*, 100273.

6. Taleb, I., Guillaume, J.-L., & Duthil, B. (2025). A survey on services placement algorithms in integrated cloud–fog/edge computing. *ACM Computing Surveys, 57*(11), Article 268.

7. Apat, H. K., Sahoo, B., Maiti, P., Barik, R. K., & Saikia, M. J. (2025). Fog service placement optimization: A survey of state-of-the-art strategies and techniques. *Computers, 14*(3), 99.

8. Supraja, T., Chawla, P., & Gill, S. S. (2025). AI-driven service placement in fog and edge computing environments: A systematic review, taxonomy and future directions. *Cluster Computing, 28*, 1070.

9. Velasquez, K., Perez Abreu, D., Curado, M., & Monteiro, E. (2021). Service placement for latency reduction in the fog using application profiles. *IEEE Access, 9*, 80821–80834.

10. Guerrero, C., Lera, I., & Juiz, C. (2018). A lightweight decentralized service placement policy for performance optimization in fog computing. *Journal of Ambient Intelligence and Humanized Computing, 9*(5), 1467–1482.

11. Guerrero, C., Lera, I., & Juiz, C. (2019). Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures. *Future Generation Computer Systems, 97*, 131–144.

12. Ghobaei-Arani, M., & Shahidinejad, A. (2022). A cost-efficient IoT service placement approach using whale optimization algorithm in fog computing environment. *Expert Systems with Applications, 200*, 117012.

13. Chikhaoui, A., Lemarchand, L., Boukhalfa, K., & Boukhobza, J. (2021). Multi-objective optimization of data placement in a Storage-as-a-Service federated cloud. *ACM Transactions on Storage, 17*(3), Article 22.

14. Liu, X., Fan, L., Wang, L., et al. (2016). Multiobjective reliable cloud storage with its particle swarm optimization algorithm. *Mathematical Problems in Engineering, 2016*, 9529526.

15. Adhikari, M., Mukherjee, M., & Srirama, S. N. (2020). DPTO: A deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing. *IEEE Internet of Things Journal, 7*(7), 5773–5782.

16. Boubaker, N. E. H., Zarour, K., Guermouche, N., & Benmerzoug, D. (2022). Fog and edge service migration approaches based on machine learning techniques. In *Proceedings of the Tunisian–Algerian Conference on Applied Computing (TACC)* (pp. 33–44). CEUR Workshop Proceedings, 3333.

17. Amazon Web Services. (2021). *What to consider when selecting a region for your workloads*. AWS Architecture Blog. https://aws.amazon.com/blogs/architecture/what-to-consider-when-selecting-a-region-for-your-workloads/.

18. FinOps Foundation. (2023). *Managing cloud cost anomalies and related FinOps capabilities*. https://www.finops.org/wg/managing-cloud-cost-anomalies/.

19. Cloudprice.net. (2025). *t4g.medium specs and pricing | AWS*. https://cloudprice.net/.

20. Bichard, L. (2021). How to find which AWS region is closest to you? *DEV Community*. https://dev.to/loujaybee/how-to-find-which-aws-region-is-closest-to-you-3k38.

Дата першого надходження статті до видання: 20.11.2025
Дата прийняття статті до друку після рецензування: 05.12.2025

UDC 004

Zinchenko Pavlo Volodymyrovych[1], Mekhed Dmytro Borysovych[2]

[1]PhD Student at the Department of Information and Computer Systems
Chernihiv Polytechnic National University (Chernihiv, Ukraine)
Email: zinchenkop@protonmail.com. **ORCID:** https://orcid.org/0009-0001-4573-6463. **ResearcherID:** MDS-7976-2025

[2]PhD in Pedagogy, Associate Professor of the Department of cybersecurity and mathematical simulation
Chernihiv Polytechnic National University (Chernihiv, Ukraine)
Email: d.mekhed@gmail.com. **ORCID:** https://orcid.org/0000-0003-3905-3620
ResearcherID: H-1751-2016. **SCOPUS Author ID:** 57193823626

INFORMATION TECHNOLOGY FOR DYNAMIC OPTIMIZATION OF MULTI-REGION CLOUD DEPLOYMENTS

The article presents an information technology aimed at the dynamic optimization of multi-region cloud deployments through the combined use of latency metrics, CDN analytics, and region-specific cloud pricing. The purpose of the work is to enhance the efficiency of cloud infrastructures by enabling automated region selection that reflects the actual geographical distribution of users and current cloud resource costs. The proposed solution includes a formal mathematical model in which effective latency is defined as a weighted minimum latency for user groups, while the total cost is calculated as the aggregated cost of active regions. An integral quality criterion is introduced to evaluate configuration alternatives.

A software agent is developed to implement the proposed approach. The agent automatically collects real-time Cloudflare analytics, including request distribution and latency measurements, as well as cloud pricing data retrieved through provider APIs. Based on the formal model, the agent periodically evaluates the active deployment configuration, identifies more optimal alternatives, and initiates reconfiguration using Infrastructure-as-Code mechanisms. A quantitative case study is provided to demonstrate the practical applicability of the approach. The results show that combining European and South Asian regions in a two-region deployment may reduce effective latency by approximately 40 %, while annual cost savings of up to USD 2800 can be achieved in scenarios with favourable cost–latency trade-offs.

The research confirms that the integration of operational latency telemetry with regional pricing information enables more efficient management of multi-region cloud systems compared to static region selection. The proposed information technology may serve as a foundation for further developments in adaptive cloud orchestration, including predictive traffic-aware scaling and integration with multi-cloud environments.

Keywords: *optimization; monitoring; latency; Cloudflare; AWS; information technology; infrastructure as code; Python; agent. Fig.: 1. References: 20.*

Зінченко П. В., Мехед Д. Б. Інформаційна технологія автоматичної оптимізації мультирегіональної інфраструктури. *Технічні науки та технології*, 2025. № 4(42). С. 208-217. DOI: https://doi.org/10.25140/2411-5363-2025-4(42)-208-217.