https://doi.org/10.31891/2219-9365-2025-82-45
УДК 004.03, 004.9

СІГУНОВ Олег
Львівський національний університет імені Івана Франка
https://orcid.org/0009-0003-6557-1429

ПОКРАЩЕННЯ ПРОДУКТИВНОСТІ ХМАРНИХ СИСТЕМ ШЛЯХОМ АДАПТИВНОЇ ОПТИМІЗАЦІЇ РЕСУРСІВ НА ОСНОВІ ГЕНЕТИЧНИХ АЛГОРИТМІВ

У статті досліджено підхід до покращення продуктивності хмарних систем шляхом адаптивної оптимізації ресурсів на основі генетичних алгоритмів (GA). Зокрема, увагу приділено оцінці ефективності системи при високих навантаженнях у гібридному хмарному середовищі AWS, що відповідає реальним умовам використання. Дослідження виконано на архітектурі, яка містить три EC2-інстанси типу t3.small, що виконували роль серверів для обробки запитів, та один EC2-інстанс типу t3.medium, який виконував роль маршрутизатора. На маршрутизаторі були реалізовані генетичні алгоритми (GA) та нейромережа (NN), яка прогнозувала пікові навантаження та допомагала адаптивно розподіляти запити.

Методологія дослідження базується на проведенні навантажувальних тестів за допомогою інструменту Gatling, що дозволяє моделювати поведінку користувачів та аналізувати продуктивність системи за різних рівнів активності. Під час тестування аналізувалися такі ключові параметри: загальний час виконання завдань, вартість використання ресурсів а також фактичне використання CPU та пам'яті. Проведена серія експериментів із різними варіантами конфігурацій, що включали використання класичного генетичного алгоритму (Classic GA), багатоцільового генетичного алгоритму (Multi-Objective GA) та гібридного алгоритму (Hybrid GA + RL) з нейромережею, яка навчалася протягом 15 хвилин, 30 хвилин, 1 години та 12 годин.

Результати дослідження продемонстрували, що використання генетичних алгоритмів суттєво покращує продуктивність системи порівняно з традиційними підходами до балансування навантаження. Особливо ефективним виявився підхід Hybrid GA + RL із тривалим навчанням нейромережі (12 годин), що забезпечив найменший час виконання завдань, оптимальне використання CPU та пам'яті, а також мінімальні витрати на ресурси серед усіх досліджуваних конфігурацій. Багатоцільовий генетичний алгоритм (Multi-Objective GA) також показав кращу продуктивність порівняно з класичним алгоритмом, особливо у випадках нестабільного навантаження.

Таким чином, отримані результати підтверджують доцільність використання адаптивної оптимізації на основі генетичних алгоритмів та нейромереж у хмарних системах AWS. Запропоновані підходи забезпечують підвищення продуктивності, зниження вартості та покращення стабільності роботи системи.

Висновки: результати досліджень можуть бути корисними для інженерів, які працюють із хмарними сервісами, а також для розробників масштабованих web-застосунків із високим навантаженням.

Ключові слова: хмарні сервіси, EC2-інстанси, Gatling, навантажувальне тестування, генетичні алгоритми, оптимізація ресурсів, нейромережі, Hybrid GA.

SIHUNOV Oleh
Ivan Franko National University of Lviv,

IMPROVING CLOUD SYSTEM PERFORMANCE THROUGH ADAPTIVE RESOURCE OPTIMIZATION BASED ON GENETIC ALGORITHMS

The article investigates an approach to improving the performance of cloud systems through adaptive resource optimization based on genetic algorithms (GA). Particular attention is paid to evaluating system efficiency under high-load conditions in a hybrid AWS cloud environment that simulates real-world usage scenarios. The study was conducted on an architecture comprising three t3.small EC2 instances, which acted as request processing servers, and one t3.medium EC2 instance that served as a router. The router hosted genetic algorithms (GA) and a neural network (NN) that predicted peak loads and helped adaptively distribute requests.

The research methodology is based on load testing using the Gatling tool, which enables user behavior simulation and system performance analysis under various load conditions. Key performance parameters such as total execution time, resource usage cost, and actual CPU and memory utilization were analyzed. A series of experiments was conducted with various configurations, including the use of the Classic Genetic Algorithm (Classic GA), the Multi-Objective Genetic Algorithm (Multi-Objective GA), and the Hybrid GA + RL algorithm with a neural network trained for 15 minutes, 30 minutes, 1 hour, and 12 hours.

The results demonstrated that using genetic algorithms significantly improves system performance compared to traditional load balancing approaches. The Hybrid GA + RL approach with 12 hours of neural network training proved to be the most effective, achieving the lowest execution time, optimal CPU and memory usage, and minimal resource costs among all tested configurations. The Multi-Objective GA also outperformed the classic algorithm, particularly in cases of unstable workloads.

Thus, the obtained results confirm the feasibility of applying adaptive optimization based on genetic algorithms and neural networks in AWS cloud systems. The proposed approaches provide enhanced performance, cost reduction, and improved system stability. The findings can be useful for engineers working with cloud services as well as developers of scalable, high-load web applications.

Keywords: cloud services, EC2 instances, Gatling, load testing, genetic algorithms, resource optimization, neural networks, Hybrid GA.

Стаття надійшла до редакції / Received 17.04.2025
Прийнята до друку / Accepted 10.05.2025

ПОСТАНОВКА ПРОБЛЕМИ У ЗАГАЛЬНОМУ ВИГЛЯДІ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

У сучасних цифрових хмарних середовищах проєкти, що працюють у режимі реального часу, вимагають ефективного використання ресурсів для забезпечення стабільної та надійної роботи. Умови непередбачуваного навантаження на інфраструктуру можуть призвести до погіршення продуктивності або надмірних витрат, тому оптимізація ресурсів стає критично важливою задачею.

Хмарні сервіси, такі як AWS, пропонують широкий спектр можливостей для гнучкого масштабування системи відповідно до змін у навантаженні [1,2]. Проте, через непередбачуваний характер споживання ресурсів стандартні методи управління часто не забезпечують оптимальної продуктивності без перевитрат ресурсів. Особливо актуальною є проблема вибору конфігурації ресурсів, яка б забезпечувала стабільну роботу системи при динамічному характері навантаження.

У хмарних середовищах для оптимізації продуктивності та стабільності необхідно враховувати різноманітні параметри, такі як обсяг доступної пам'яті, рівень завантаження CPU та можливі затримки у відповіді системи [3]. Крім того, гнучкість інфраструктури є ключовим фактором при роботі з системами, що мають пікові навантаження або непередбачувані сплески активності.

Генетичні алгоритми є ефективним інструментом для динамічної адаптації конфігурації ресурсів у хмарних середовищах [4,5]. Вони імітують природні механізми еволюції, що дозволяє швидко знаходити оптимальні рішення у складних динамічних системах. У даній статті розглядається застосування генетичних алгоритмів для оптимізації розподілу навантаження між трьома EC2-інстансами. Для управління маршрутизацією запитів між інстансами використано окремий EC2-інстанс, який виконує роль маршрутизатора та виконує обчислення для генетичних алгоритмів. Для оцінки продуктивності системи проведено навантажувальне тестування за допомогою інструмента Gatling. Показано, що використання генетичних алгоритмів дозволяє значно підвищити продуктивність системи, зменшити витрати ресурсів та забезпечити стабільність роботи навіть за раптових сплесків активності користувачів.

Обгрунтування та вибір середовища розгортання тестової системи

Вибір середовища розгортання для системи, що використовує генетичні алгоритми для оптимізації розподілу ресурсів, потребує врахування особливостей роботи в умовах високих навантажень та адаптивного балансування трафіку. З огляду на це, оптимальним рішенням є використання хмарного середовища Amazon Web Services (AWS), що забезпечує гнучкість у налаштуванні інфраструктури та підтримку необхідних сервісів для моделювання реальних умов експлуатації.

У якості середовища розгортання було обрано EC2-інстанси завдяки їхній здатності забезпечити повний контроль над конфігурацією системи, що є важливим для коректної роботи генетичних алгоритмів та нейромережі. Перевагою EC2 є можливість налаштування інстансів під конкретні обчислювальні вимоги, що дозволяє досягти оптимального співвідношення між вартістю ресурсів та їх продуктивністю.

Для тестування ефективності системи та її роботи під навантаженням використано інструмент Gatling, що дозволяє імітувати великий обсяг трафіку від користувачів та аналізувати поведінку системи при різних рівнях активності. Gatling надає детальні звіти про продуктивність системи, включаючи час обробки запитів, стабільність роботи та рівень використання ресурсів.

Запропонована конфігурація забезпечує гнучкість, високу продуктивність та стабільність системи в умовах змінних навантажень, що дозволяє ефективно застосовувати генетичні алгоритми для оптимізації розподілу ресурсів у хмарному середовищі AWS.

Перелік та особливості використаних технологій

Scala — це сучасна мова програмування, яка працює на платформі Java Virtual Machine (JVM). Scala підтримує модель акторів, що дозволяє визначати об'єкти як незалежні сутності з власними властивостями та поведінкою. Це покращує взаємодію між потоками та підвищує ефективність обробки даних.

Gatling — це програмне забезпечення для навантажувального тестування, написане на Scala. Воно дозволяє симулювати великий потік трафіку та оцінювати, як система реагує на різні рівні навантаження. Gatling допомагає виявити вузькі місця та покращити стабільність системи.

Для оцінки ефективності системи застосовувалося тестування продуктивності (Performance Testing). У рамках цього тестування аналізувалися показники часу виконання завдань, ефективності використання CPU/пам'яті та загальних витрат на ресурси.

Для нашого дослідження застосовувалася стратегія constantConcurrentUsers, яка дозволяє моделювати постійне навантаження від одночасних користувачів.

Параметри оцінки продуктивності включають:

- Загальний час виконання завдань (Execution Time)
- Загальна вартість використання ресурсів (Cost)
- Фактичне використання CPU (Actual CPU Utilization)
- Фактичне використання пам'яті (Actual Memory Utilization)

Gatling також надає детальні звіти для аналізу отриманих результатів.

AWS EC2 — це віртуальні сервери, що використовувалися для обробки запитів у нашій системі. Було розгорнуто три EC2-інстанси типу t3.small для обробки навантаження та один EC2-інстанс типу t3.medium для маршрутизації трафіку та виконання генетичних алгоритмів (GA).

AWS Elastic Load Balancer (ELB) використовується для рівномірного розподілу вхідного трафіку між EC2-інстансами, що забезпечує стабільність роботи системи навіть під високим навантаженням.

AWS Relational Database Service (RDS) використовувалася як основна база даних для трьох EC2-інстансів, що обробляли запити. RDS забезпечувала надійність, стабільність та швидке зчитування даних під високим навантаженням.

Структура проєкту та його розміщення у хмарному середовищі

Розгортання системи для тестування було реалізовано за допомогою Amazon Web Services (AWS), враховуючи його гнучкість, надійність та потужний інструментарій для автоматизації масштабування та моніторингу ресурсів. Вибір AWS обумовлений його широким спектром сервісів, що дозволяють ефективно керувати обчислювальними ресурсами та оптимізувати витрати на інфраструктуру.

У системі використано чотири EC2-інстанси: три EC2-інстанси типу t3.small, що виконують роль серверів для обробки запитів, та один EC2-інстанс типу t3.medium, що виконує роль маршрутизатора та відповідає за виконання генетичних алгоритмів (GA) та нейромережі (NN). Такий підхід дозволяє розділити навантаження між окремими інстансами, що сприяє кращій стабільності системи та зменшенню затримок у обробці запитів.

Використання інстансів типу t3.small обумовлено їх економічною доцільністю, оскільки вони забезпечують оптимальний баланс між вартістю та продуктивністю, що важливо для сценаріїв із нерівномірним або сплесковим навантаженням. Інстанс типу t3.medium завдяки збільшеним ресурсам CPU та пам'яті забезпечує стабільну роботу алгоритмів оптимізації навіть під час високого навантаження.

Для зберігання даних застосовано сервіс AWS RDS, що виконує функцію бази даних. Використано сервер типу db.t3.medium, який забезпечує баланс між продуктивністю та вартістю. Вибір такого типу бази даних обумовлений необхідністю виключити вплив продуктивності бази даних на результати тестування. RDS забезпечує автоматичне резервне копіювання, що гарантує цілісність і доступність даних у разі аварійних ситуацій. Такий підхід дозволяє зосередитися на тестуванні саме продуктивності EC2-інстансів та оптимізації їх роботи за допомогою генетичних алгоритмів.

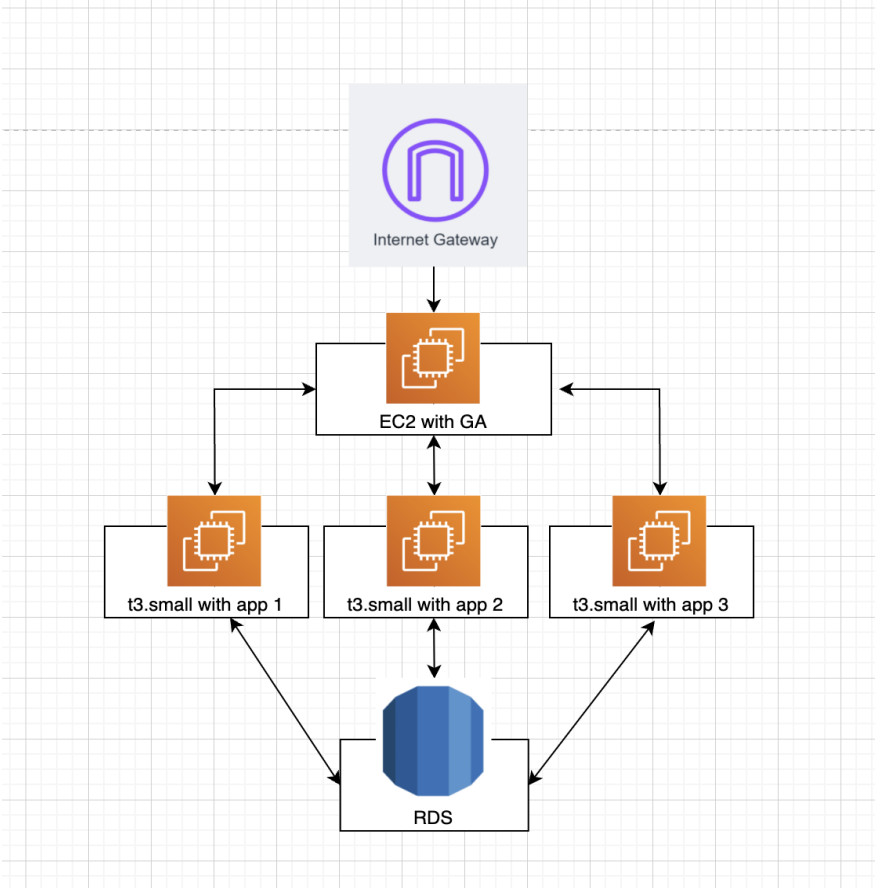

Рис. 1. Схема розгортання web-застосунку для тестування

Архітектура системи зображена на рисунку 1. На схемі показано, як трафік від користувачів надходить до EC2-інстансу з GA, який здійснює маршрутизацію запитів між іншими EC2-інстансами для забезпечення рівномірного навантаження. При цьому враховується поточне завантаження кожного інстансу та прогнозоване навантаження на основі даних нейромережі. База даних AWS RDS отримує та зберігає необхідні дані про виконання завдань та користувачів.

Контейнеризація в межах цього дослідження не застосовувалася, оскільки система базується на віртуальних серверах EC2 для досягнення максимальної близькості до умов, які виникають при реальних навантаженнях на сервери програм. Такий підхід дозволяє точно оцінити ефективність роботи генетичних алгоритмів та проаналізувати їхній вплив на продуктивність системи в умовах динамічного навантаження.

Тестування системи

Теоретичний опис методики тестування

Для оцінки продуктивності системи було розроблено тестовий сценарій, що максимально наближений до реальних умов експлуатації. Система імітувала ситуацію, коли користувачі надсилали запити на сервер для отримання даних у реальному часі.

Кількість одночасних користувачів була встановлена на рівні 50 для оцінки впливу навантаження на систему та її стабільності при різних рівнях активності. Кожен користувач після отримання відповіді негайно повторно відправляв запит на сервер, що дозволяло відтворити умови постійного навантаження.

Тестування виконувалося для обробки фіксованого обсягу запитів, що дозволило оцінити стабільність системи та її продуктивність у реальних умовах. Такий підхід дозволив не лише оцінити працездатність системи в реальних умовах, але й перевірити, як вона поводиться у випадку пікових навантажень, що є важливим для оцінки ефективності роботи генетичних алгоритмів.

Для обчислення вартості ресурсів було враховано тривалість роботи EC2-інстансів та їхній тип. Вартість розраховувалася на основі стандартних тарифів AWS, що дозволило об'єктивно оцінити витрати на кожен з підходів до оптимізації.

Під час тестування збиралися такі параметри:

- Загальний час виконання завдань (Execution Time)
- Загальна вартість використання ресурсів (Cost)
- Фактичне використання CPU (Actual CPU Utilization)
- Фактичне використання пам'яті (Actual Memory Utilization)

Генетичні алгоритми були реалізовані за допомогою бібліотеки Pygad, яка забезпечує простий інтерфейс для створення та налаштування генетичних алгоритмів. Ця бібліотека була обрана завдяки її гнучкості та легкості використання, що дозволило швидко адаптувати алгоритми під специфіку тестованої системи.

Для оцінки ефективності були застосовані три типи генетичних алгоритмів:

- Класичний генетичний алгоритм (Classic GA) — базова реалізація без додаткових оптимізацій.
- Гібридний алгоритм GA + RL (Hybrid GA + RL) — комбінація генетичного алгоритму та алгоритмів з підкріпленням для адаптивного коригування конфігурації системи.
- Багатоцільовий генетичний алгоритм (Multi-Objective GA) — алгоритм, що одночасно оптимізує кілька параметрів, таких як продуктивність і вартість.

Результати тестування

Проведені експерименти показали значні відмінності між результатами різних підходів до оптимізації системи.

У таблиці 1 подано результати тестування системи з використанням двох типів генетичних алгоритмів (**Classic GA** та **Multi-Objective GA)**.

Таблиця 1

Результати тестування системи, з використанням алгоритмів Classic GA та Multi-Objective GA

Алгоритм	Загальний час виконання (с)	Загальна вартість ($)	Використання CPU (%)	Використання пам'яті (GB)
Classic GA	1180	0.0575	65.2	93
Multi-Objective GA	1035	0.0547	79.5	101

У таблиці 2 подано результати дослідження системи при використанні генетичного алгоритму Hybrid GA + RL.

Таблиця 2

Вплив часу навчання нейромережі на продуктивність системи при використанні алгоритму Hybrid GA + RL.

Алгоритм	Загальний час виконання (с)	Загальна вартість ($)	Використання CPU (%)	Використання пам'яті (GB)
Hybrid GA + RL (15 хв)	1280	0.059	72.4	102
Hybrid GA + RL (30 хв)	1180	0.056	75.5	107
Hybrid GA + RL (1 год)	1080	0.055	78.3	110
Hybrid GA + RL (12 год)	950	0.053	85.5	104

Результати тестування свідчать про значну перевагу використання нейромережі в поєднанні з генетичними алгоритмами за умов її тривалого навчання. Hybrid GA + RL із 12 годинами навчання показав найкращий баланс між продуктивністю, витратами та ефективністю використання ресурсів.

Для наочного порівняння продуктивності системи при використанні Hybrid GA + RL із різним часом навчання було створено чотири графіки, які відображають ключові параметри роботи системи (рис.2-рис.4)

На графіку спостерігається поступове зниження часу виконання завдань зі збільшенням тривалості навчання нейромережі. Найпомітніше покращення відбувається після 12 годин навчання, коли загальний час виконання завдання знижується до 950 секунд, що на ~25% швидше за варіанту із 15 хвилинами навчання.

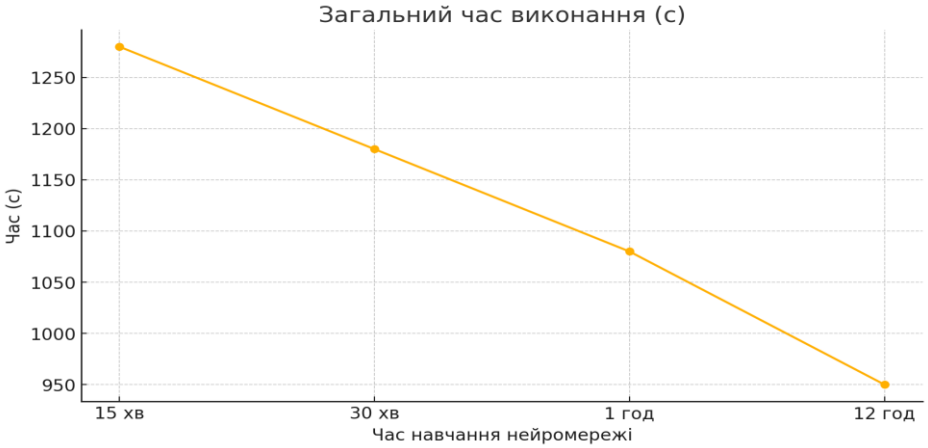

Рис. 2. Залежність часу виконання завдання від часу навчання нейромережі

Загальна вартість використання ресурсів поступово зменшується із покращенням продуктивності системи. Після 12 годин навчання вона знижується до 0.053 $, що є найнижчим показником серед усіх варіантів.

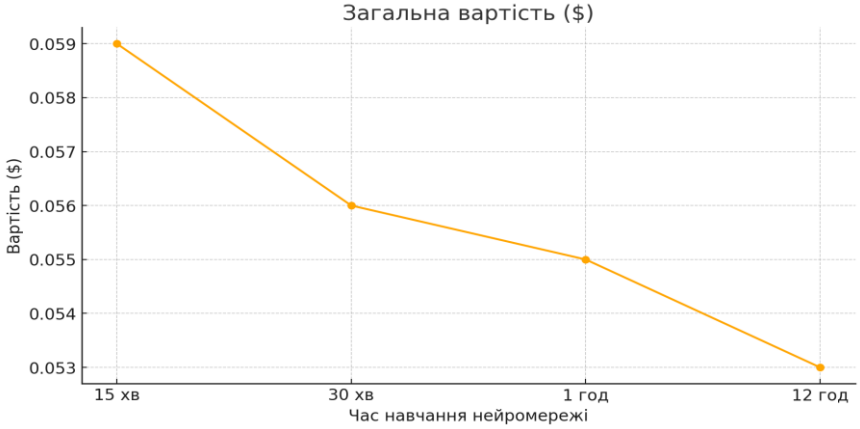

Рис. 3. Залежність загальної вартості використання ресурсів від часу навчання нейромережі

Відзначається стабільне зростання рівня використання CPU зі збільшенням часу навчання нейромережі. Це пояснюється тим, що після тривалішого навчання нейромережа краще оптимізує розподіл ресурсів, завантажуючи їх більш рівномірно та ефективно.

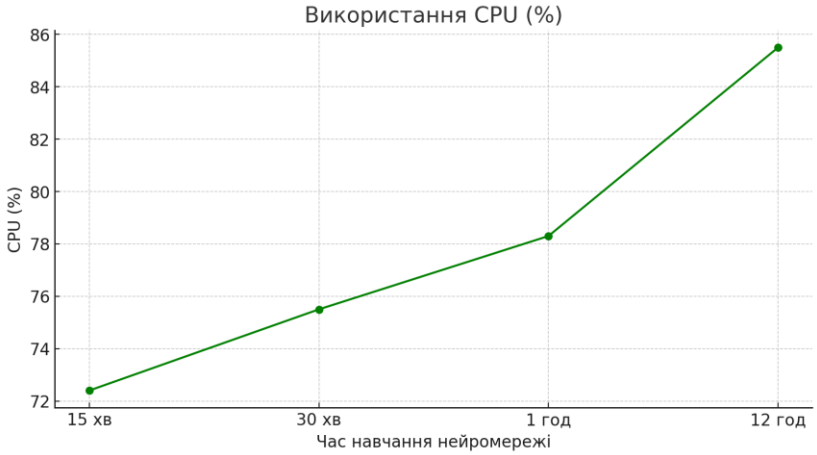

Рис. 4. Залежність використання CPU від часу навчання нейромережі

Використання пам'яті демонструє коливання на всіх етапах навчання. Після 12 годин навчання рівень використання пам'яті дещо знижується (до 104 GB) порівняно з проміжними варіантами.

Рис. 5. Залежність використання пам’яті від часу навчання нейромережі

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

За результатам виконання роботи було створено архітектуру системи з використанням EC2-інстансів, яка застосовує різні підходи до оптимізації розподілу ресурсів за допомогою генетичних алгоритмів. Під час тестування було оцінено продуктивність системи при використанні Classic GA, Multi-Objective GA та Hybrid GA + RL із різним часом навчання нейромережі.

Проведене тестування показало, що Multi-Objective GA демонструє кращі показники продуктивності та економічності порівняно з Classic GA. Цей алгоритм дозволив зменшити як загальний час виконання завдань, так і витрати на використання обчислювальних ресурсів.

Найкращі результати було досягнуто за допомогою Hybrid GA + RL із тривалим навчанням нейромережі (12 годин). Цей підхід продемонстрував найнижчий час виконання завдань, оптимальне використання CPU та пам'яті, а також мінімальні витрати серед усіх протестованих конфігурацій.

Таким чином, отримані результати свідчать про значну перевагу використання нейромереж у поєднанні з генетичними алгоритмами для оптимізації продуктивності систем у хмарних середовищах. Для реалізації подібних завдань доцільно застосовувати Hybrid GA + RL із тривалим навчанням нейромережі як найоптимальніший підхід для досягнення високої продуктивності та економії ресурсів.

На основі отриманих результатів можна зробити висновок, що тривале навчання нейромережі в поєднанні з генетичним алгоритмом забезпечує суттєве покращення продуктивності системи, зниження витрат на ресурси та більш ефективне використання обчислювальних потужностей. Комбінація Hybrid GA + RL із 12 годинами навчання демонструє найкращі результати серед досліджуваних варіантів.

Проведені експерименти показали значні відмінності між різними підходами до оптимізації системи.

ISSN 2219-9365

References

[1] Goldberg, D.E. Genetic Algorithms in Search, Optimization, and Machine Learning // Addison-Wesley, 1989. ISBN: 978-0201157673.

[2] Mitchell, M. An Introduction to Genetic Algorithms // MIT Press, 1998. ISBN: 978-0262631853.

[3] Whitley, D. A Genetic Algorithm Tutorial // *Statistics and Computing*, 4(2):65-85, 1994. https://doi.org/10.1007/BF00175354

[4] Holland, J.H. Adaptation in Natural and Artificial Systems // University of Michigan Press, 1975. ISBN: 978-0262581110.

[5] Moscato, P., Cotta, C. A Gentle Introduction to Memetic Algorithms // *Handbook of Metaheuristics*, 2003. https://doi.org/10.1007/978-1-4757-4217-7_29

[6] Ranaweera, S., Fung, C., Kulkarni, R. Optimizing Cloud Resource Allocation with Genetic Algorithms // *Journal of Cloud Computing: Advances, Systems and Applications*, 9(1), 2021. https://doi.org/10.1186/s13677-021-00263-9

[7] Lin, S., Zhang, X., Chen, Y. Adaptive Load Balancing with Genetic Algorithms in Cloud Computing // *Future Generation Computer Systems*, 95:424-434, 2019. https://doi.org/10.1016/j.future.2019.10.018

[8] Tian, W., Xu, Y., Sun, H. Performance Analysis of Cloud Systems Using Neural Networks and Genetic Algorithms // *Journal of Systems and Software*, 159:110435, 2020. https://doi.org/10.1016/j.jss.2019.110435