УДК 004.4 https://doi.org/10.32515/2664-262X.2025.12(43).1.44-57

О. С. Улічев[1], канд. техн. наук, **В. П. Кулагін**[2]
[1]*Центральноукраїнський національний технічний університет м. Кропивницький, Україна*
[2]*Приватний вищий навчальний заклад "Європейський університет", м. Київ, Україна*
e-mail: askin79@gmail.com, victor@kulagin.com.ua

Оптимізація роботи мікросервісів на основі теорії ігор

Завдяки високій гнучкості та здатності до масштабування мікросервісна архітектура стала актуальним і популярним підходом у розробці сучасних програмних систем. Водночас її застосування ставить нові завдання, пов'язані з керуванням ресурсами та забезпеченням стабільної якості обслуговування. У цій статті запропоновано підхід до оптимізації роботи мікросервісів на основі теорії ігор. Запропоновано модель, в якій кожен мікросервіс розглядається як гравець, що прагне оптимізувати власну вигоду (наприклад, мінімізувати час відповіді) при спільному використанні обмежених ресурсів. Використовуючи концепцію рівноваги Неша, розроблено Game Theory Controller - контролер, що обчислює оптимальний розподіл ресурсів між мікросервісами. Наведено математичну модель такої гри, псевдокод алгоритму пошуку рівноваги, а також архітектурну схему інтеграції контролера в систему оркестрації контейнерів. Експериментальні результати моделювання демонструють, що запропонований підхід дозволяє збалансувати навантаження між сервісами та зменшити середній час відповіді системи у порівнянні з традиційними стратегіями (наприклад, Binpack та Spread). У висновках обговорено доцільність застосування теорії ігор для керування мікросервісами та окреслено напрями подальших досліджень.
мікросервісна архітектура, оптимізація, теорія ігор, рівновага Неша, розподіл ресурсів, відмовостійкість, оркестрація, якість обслуговування (QoS)

Постановка проблеми. Перехід від монолітної архітектури до набору мікросервісів породжує ряд проблем. Зокрема, керування ресурсами (процесорним часом, пам'яттю, пропускною здатністю мережі тощо) у середовищі з великою кількістю сервісів стає складнішим завданням. Кожен сервіс має власні вимоги до ресурсів і показників якості обслуговування (Quality of Service, QoS), а ресурси інфраструктури обмежені. Якщо ресурси розподілено неефективно, може погіршитися час відповіді сервісів, зрости кількість відмов запитів або порушитися угоди про рівень сервісу (Service Level Agreement, SLA). Традиційні підходи до керування ресурсами у хмарних середовищах (наприклад, статичне виділення ресурсів, евристики на кшталт Binpack та Spread для розміщення контейнерів) не завжди забезпечують оптимальну продуктивність для мікросервісних застосунків. Відтак постає проблема динамічної оптимізації розподілу ресурсів між мікросервісами з урахуванням їх поточного стану та навантаження. Необхідно забезпечити, щоб жоден сервіс не страждав від дефіциту ресурсів, коли інші простоюють, і водночас уникнути ситуації, коли всі сервіси одночасно агресивно споживають ресурси, заважаючи один одному.

Таким чином, формулювання проблеми зводиться до наступного: як координувати розподіл спільних обчислювальних ресурсів між незалежними мікросервісами, які одночасно співпрацюють задля виконання загальної бізнес-логіки та конкурують за ресурси, з метою оптимізації глобальних показників системи (пропускної здатності, середнього часу відповіді, кількості оброблених запитів) при дотриманні локальних цілей кожного сервісу, зокрема їхніх QoS-вимог. Цю задачу пропонується вирішувати із залученням апарату теорії ігор, математичного інструментарію для аналізу поведінки множини раціональних агентів (гравців), інтереси яких частково

співпадають, а частково суперечать один одному.

Аналіз останніх досліджень і публікацій. Мікросервісна архітектура характеризується розподілом застосунку на набір дрібних незалежно розгорнутих сервісів, що спілкуються між собою через чітко визначені інтерфейси. Такий підхід дозволяє спростити розробку та масштабування складних систем, підвищити стійкість до збоїв та прискорити випуск оновлень. У попередній статті було розглянуто мікросервісну архітектуру, як сучасний підхід до розробки програмного забезпечення, який відповідає щоразу вищим вимогам бізнесу та технологій, розглянуто її переваги та недоліки в порівнянні з іншими архітектурами [1]. За даними досліджень, мікросервіси набули значного поширення в індустрії програмного забезпечення завдяки можливості горизонтального масштабування та ізоляції компонентів [2]. Протягом останніх десятиліть теорія ігор демонструвала ефективність у розв'язанні задач оптимального розподілу ресурсів в обчислювальних системах. Ще з 2000-х років науковці активно використовували ігрові моделі для опису процесів керування мережами та дата-центрами, де окремі вузли або користувачі розглядаються як гравці, що прагнуть покращити власні показники за допомогою вибору стратегій. E. Altman та інші [3] навели огляд застосування теорії ігор у телекомунікаційних мережах, продемонструвавши ефективність граничної рівноваги в задачах керування трафіком та мережевих ресурсів. У сфері хмарних обчислень одним із перших прикладів використання теорії ігор є дослідження G. Wei та співавторів [4], де було запропоновано ігрову модель для справедливого розподілу обчислювальних ресурсів між користувачами. Такий підхід дозволив враховувати інтереси всіх сторін і досягати рівноваги Неша.

У свою чергу, D. Ardagna з колегами [5] використали узагальнену модель рівноваги Неша для вирішення задач, пов'язаних із наданням хмарних сервісів, де кілька провайдерів одночасно прагнули підвищити власний прибуток, не знижуючи рівень обслуговування клієнтів. Ці дослідження заклали основу для використання ігрових моделей у системах розподілених обчислень.

З поширенням контейнеризації та мікросервісів науковці почали звертати увагу на специфічні виклики, пов'язані з динамічним характером таких систем. Керування ресурсами на рівні контейнерів (microservice resource management) суттєво відрізняється від традиційного керування віртуальними машинами, оскільки одиниць розгортання (контейнерів/сервісів) набагато більше, а їх життєвий цикл коротший. У ряді праць було запропоновано застосувати теоретико-ігрові методи безпосередньо до середовища мікросервісів та fog/edge-інфраструктури (обчислення на периферії мережі). Наприклад, K. Kaur та інші [7] сконструювали багатокритерійну функцію корисності для сервісів на основі моделі кооперативної гри, щоб збалансувати енергоспоживання та доступність сервісів у так званих нано-центрах обробки даних на краю мережі. У їхньому підході контейнери з мікросервісами розгортаються на вузлах туманних обчислень (fog computing), а сервіс вибирає стратегію, що оптимізує одночасно час виконання завдань і енергоспоживання пристроїв, досягаючи компромісу між ефективністю та надійністю роботи.

Інший приклад - S. Yan та співавтори [8], які розглянули гру між користувачами та точками доступу у туманній радіомережі (Fog Radio Access Network, F-RAN). Вони запропонували еволюційну гру для вибору режиму доступу: користувачі вирішують, відправляти трафік безпосередньо на базову станцію чи на локальний вузол обчислень, на основі виграшу в пропускній здатності та затримці. Встановлено, що така гра сходиться до еволюційної рівноваги, яка максимізує ефективність використання мережевих і обчислювальних ресурсів F-RAN. Хоча ця робота стосується мережевого

рівня, вона демонструє підхід, коли множина агентів (користувачів) самоорганізується через локальні рішення в глобально оптимальний або наближений до оптимального стан.

Безпосередньо до проблем оптимізації мікросервісних runtime-середовищ теорію ігор застосували R. Luo та інші [9]. Вони відзначили, що мікросервіси в межах одного застосунку співпрацюють для обслуговування запитів (через виклики API один одного), але водночас можуть конкурувати за спільні ресурси сервера або кластера, особливо при піковому навантаженні. Автори змоделювали цю ситуацію як конгестивну гру (congestion game): кожен мікросервіс (гравець) прагне максимізувати свій дохід, який визначається рівнем дотримання вимог SLA (наприклад, штрафи за перевищення часу відповіді). Якщо надто багато сервісів одночасно споживають один і той самий ресурс (CPU, пам'ять), їхній індивідуальний виграш зменшується через ріст часу обробки - це явище «конгестії» (перевантаження). У моделі R. Luo для кожного сервісу визначено функцію корисності (доходу) θ_i, що лінійно спадає зі збільшенням часу відповіді r_i на його запити:

$$\theta_i = v_i + m_i r_i, m_i < 0 \quad (1)$$

де v_i - базовий дохід (коли час відповіді прямує до нуля), m_i - коефіцієнт "штрафу" за затримку (від'ємне значення), а r_i - час відповіді.

Таким чином, чим довше час відповіді сервісу i, тим менше θ_i, тобто сервіс зацікавлений мінімізувати власний час обслуговування. Весь набір сервісів формує гру, де стратегія кожного гравця полягає у виборі деякого розподілу своїх запитів по наявних ресурсах (наприклад, по різних вузлах кластера). R. Luo та інші довели, що ця гра належить до класу потенційних ігор (potential games), для яких гарантовано існування щонайменше однієї чистої рівноваги Неша. Вони запропонували алгоритм поліноміального часу для знаходження такої рівноваги, фактично реалізований у вигляді компонента-контролера, що збирає інформацію про поточні навантаження сервісів і рекомендує їм оптимальний розподіл запитів між ресурсами. Експериментально показано, що їхній підхід перевершує за ефективністю стандартні політики Docker Swarm (Binpack і Spread): середній час відповіді запитів зменшився, а кількість відмов (через тайм-аут) - теж суттєво нижча при використанні ігрового розподілу ресурсів [9].

На додачу до згаданих робіт, є низка інших досліджень, де теорію ігор застосовано для споріднених завдань оптимізації в edge/cloud-системах. Q. He та співавтори [10] розробили ігрову модель для розподілу користувачів між кількома крайовими вузлами, щоб максимізувати виграш як користувачів, так і провайдерів послуг на краю мережі. Їхній підхід дозволяє динамічно залучати нові крайові сервери або вивільняти існуючі, встановлюючи рівновагу між навантаженням і вартістю обслуговування.

S. Kumar [11] з колегами сфокусувалися на багатоорендарних (multi-tenant) середовищах на краю інтернету речей: вони застосували некооперативну гру для ціноутворення ресурсів, аби стимулювати провайдерів edge-хмар підвищувати коефіцієнт використання своїх серверів без шкоди для QoS сервісів. Розв'язання задачі Edge Resource Allocation у них досягається через обчислення рівноважних цін на ресурси, за яких жоден провайдер і жоден споживач (сервіс) не виграє від односторонньої зміни своєї стратегії (ціни чи обсягу ресурсів). Варто відзначити, що

теорія ігор застосовується не лише для оптимізації продуктивності та витрат, але й для забезпечення надійності та безпеки в мікросервісних системах. Зокрема, існують роботи, де ігрові моделі використовуються для протидії кібератакам (наприклад, розподіленим атакам типу відмова в обслуговуванні на краю мережі) шляхом знаходження рівноважних стратегій між атакувальниками ти захисними мікросервісами [12]. Проте в контексті даної статті основну увагу приділено саме оптимізації розподілу обчислювальних ресурсів і забезпеченню якості обслуговування.

Окрім підходів на основі теорії ігор, в літературі запропоновано й інші методи оптимізації мікросервісних архітектур. Так, K. Velasquez та інші [14] досліджували задачу розміщення сервісів (service placement) в інфраструктурі Інтернету речей. Вони мінімізували затримку мережі між датчиками та сервісами шляхом вибору оптимальних вузлів для їх розміщення. Хоча їхній метод зменшує кількість «стрибків» (hop count) в мережі та тим самим затримку, він не враховує обмеженість обчислювальних ресурсів на вузлах. Це може призвести до перевантаження певних вузлів при великій кількості сервісів.

L. Baresi з колегами [15] запропонували підхід, заснований на теорії керування: вони розробили дискретний регулятор зі зворотним зв'язком, який динамічно змінює кількість запущених контейнерів з мікросервісами або виділені їм ресурси, виходячи з відхилення показників QoS від цільових значень. Такий підхід забезпечує стабілізацію часу відповіді сервісів, проте вимагає ретельного налаштування контролера та моделей системи.

C. Guerrero та інші [16] застосували методи метаевристичної оптимізації (генетичні алгоритми) для розподілу контейнерів у мультихмарному середовищі: їхня мета зменшити час реакції застосунку та вартість оренди хмарних ресурсів шляхом знаходження майже оптимального розподілу сервісів між декількома хмарними провайдерами. Подібні підходи можуть досягати хороших результатів, але зазвичай потребують значного обчислювального часу для пошуку рішення і не гарантують глобальної оптимальності. Попередні дослідження продемонстрували, що теорія ігор може бути дієвим інструментом для оптимізації різних аспектів роботи розподілених систем, від хмарних технологій до туманних обчислень і мікросервісної архітектури. Водночас, проблема цілісного управління ресурсами в межах мікросервісної системи (на рівні контейнерів і сервісної оркестрації) залишається відкритою і потребує подальшого дослідження. Лише декілька праць безпосередньо адресують конкурентний розподіл ресурсів між мікросервісами одного застосунку. Необхідне рішення, яке б у режимі реального часу балансувало потреби багатьох сервісів та приводило систему до стану, близького до соціального оптимуму (оптимуму для всього застосунку), враховуючи при цьому егоїстичні інтереси окремих сервісів. Саме тому в даній статті ставиться задача розробити підхід, що використовує теорію ігор для оптимізації мікросервісної архітектури, та продемонструвати його реалізацію і переваги.

Постановка завдання. Метою роботи є підвищення ефективності функціонування мікросервісної архітектури шляхом розробки ігрової моделі розподілу обчислювальних ресурсів між сервісами та відповідного контролера для її реалізації. Для досягнення цієї мети необхідно вирішити наступні завдання.

1. Формалізувати ситуацію спільного користування ресурсами як некооперативну гру: визначити множину гравців (мікросервісів), їхні стратегії (параметри використання ресурсів, наприклад, доля CPU або кількість інстансів, що запускаються), та функції виграшу (утиліти), які відображають якість роботи сервісів. Модель має враховувати як колаборативні аспекти (ланцюжки викликів між сервісами, спільне виконання запиту), так і конфліктні (конкуренція за CPU, пам'ять).

2. Обґрунтувати, що оптимальний на думку кожного окремого сервісу розподіл ресурсів (стратегія) приводить систему до рівноваги Неша, стану, в якому жоден сервіс не може поліпшити свій показник, односторонньо змінивши стратегію [9]. В ідеалі цей стан має відповідати близькому до глобального оптимуму розподілу ресурсів. Слід дослідити існування та єдиність рівноваги, використовуючи, наприклад, апарат потенційних ігор [13].

3. Розробити ефективний алгоритм, що знаходить (або наближує) рівноважний розподіл ресурсів між сервісами. Це може бути ітераційний процес покращення стратегій (так званий алгоритм найкращої відповіді), або ж пряме розв'язання оптимізаційної задачі, еквівалентної грі. Важливо забезпечити збіжність алгоритму за прийнятне число кроків навіть при великій кількості сервісів.

4. Спроєктувати компонент Game Theory Controller - програмний модуль, що інтегрується в існуючу систему оркестрації (наприклад, у Kubernetes або Docker Swarm) і виконує функції:

– збір метрик роботи сервісів (частота запитів, час відповіді, використання CPU/пам'яті);

– вирішення ігрової моделі (обчислення нової рівноважної конфігурації);

– застосування змін (масштабування сервісів, перенаправлення навантаження, встановлення лімітів ресурсів).

Необхідно продумати, як контролер взаємодіятиме з оркестратором через його API, і забезпечити, щоб ця взаємодія не створювала великого оверхеду.

5. Змоделювати або реалізувати прототип системи з Game Theory Controller та порівняти його поведінку з базовими стратегіями (без оптимізації). Для достовірності бажано використати сценарії різного навантаження та конфігурацій сервісів. Показниками ефективності мають бути середній час відповіді, пропускна здатність (кількість успішно оброблених запитів за секунду), кількість порушень SLA тощо.

Результатом вирішення цих завдань має стати науково обґрунтований та перевірений підхід до оптимізації мікросервісної архітектури, який демонструє переваги над існуючими евристичними підходами.

Виклад основного матеріалу. Розглянемо застосунок, реалізований у вигляді набору мікросервісів $S = 1,2,...,n$, що працюють у спільному оточенні (кластері контейнерів). Нехай маємо скінченну множину ресурсів $R = 1,2,...,R$, наприклад, обчислювальні вузли або абстрактні ресурси на кшталт CPU-часу. Кожен мікросервіс $i \in S$ отримує потік запитів із певною інтенсивністю λ_i (залежить від загального вхідного навантаження та бізнес-логіки). Якщо ресурсів достатньо, сервіс обробляє запити з середнім часом r_i , який зростає при перевантаженні. Мікросервіси можуть викликати один одного: припустимо, існує певний граф викликів між сервісами (наприклад, сервіс автентифікації викликається кількома іншими сервісами тощо). Проте для спрощення моделі припустимо, що ключовий вплив на продуктивність сервісу мають виділені йому ресурси, а не черги викликів - це вплив можна врахувати у функції корисності як ваговий коефіцієнт важливості сервісу.

Запропонуємо функцію корисності (виграшу) для мікросервісу i :

$$\theta_i(x_{i1}, x_{i2}, \ldots, x_{iR}) \tag{2}$$

показник ефективності роботи сервісу i , що залежить від розподілу ресурсів x_{iR} , які він отримує від кожного ресурсу $i \in S$.

Ця функція повинна відображати, наскільки добре сервіс i задовольняє свої SLA-вимоги. Один із варіантів, аналогічно до роботи R. Luo [9] - визначити θ_i через рівень виконання сервісом i цільового часу відповіді.

1. Встановимо бажаний максимальний час відповіді $T_i^{(SLA)}$ для сервісу i згідно SLA.

2. Нехай фактичний середній час відповіді r_i є функцією від виділеного ресурсу (або ресурсів). Якщо сервіс отримує більше CPU або більше екземплярів запущено, r_i зменшується.

3. Оцінимо виграш як штраф за перевищення часу:

$$\theta_i = \begin{cases} 1, \text{ якщо } r_i \le T_i^{(SLA)} \\ \dfrac{T_i^{(SLA)}}{r_i}, \text{якщо } r_i > T_i^{(SLA)} \end{cases} \quad (3)$$

$\theta_i = 1$, якщо SLA виконується, і менше 1, якщо сервіс працює повільніше пропорційно до погіршення.

Інший підхід - задати θ_i як дохід сервісу, який він приносить, з урахуванням штрафів за поганий QoS [9]. Наприклад, дохід сервісу i може бути пропорційним кількості успішно оброблених запитів (що обернено пропорційна до r_i) мінус штрафи за кожен запит, що перевищив допустимий час або був втрачений.

Незалежно від конкретної форми θ_i, властивість буде спільною: θ_i монотонно спадає зі зменшенням виділених сервісу ресурсів (або з ростом навантаження при фіксованих ресурсах). Мікросервіс зацікавлений максимізувати θ_i.

Тепер формулюємо гру:

$$G = \langle S, \sum_i i \in S, \theta_i i \in S \rangle \quad (4)$$

гравці: мікросервіси $i \in S$, стратегії: $\sum_i$ - множина можливих стратегій сервісу i.

Стратегія визначає, як сервіс розподіляє своє навантаження або запити по наявних ресурсах. Наприклад, якщо є R однакових вузлів, стратегія може задавати число інстансів сервісу i на кожному з вузлів (вектор $(x_{i1}, \ldots, x_{iR})$, де $x_{ir} \ge 0$ і $\sum_r x_{iR} = N_i$ - загальне число контейнерів сервісу i). Або стратегія - це вибір одного з вузлів, на якому виконуватиметься сервіс (тоді $\sum_i$ - дискретна множина вузлів). У випадку спільного CPU-часу стратегія може бути безперервною: який пріоритет або долю CPU вимагати.

Функція виграшу:

$$\theta_i(\sigma_i, \sigma_{-i}) \quad (5)$$

де $\sigma_i \in \sum_i$ - стратегія гравця i, а σ_{-i} - профіль стратегій всіх інших.

Виграш i залежить не лише від його власної стратегії, а й від стратегій інших, бо вони впливають на загальне навантаження кожного ресурсу. Наприклад, якщо сервіси 1 і 2 розміщені на одному вузлі, їхній час відповіді зросте для обох через спільне використання CPU - це і є взаємодія стратегій.

У багатьох задачах такого типу зручно розглядати потенційну функцію $P(\sigma_1,...,\sigma_n)$, зміна якої від стратегії окремого гравця пропорційна зміні його виграшу [13]. Інтуїтивно, потенційна функція тут може відображати «загальне задоволення системи» розподілом ресурсів. Якщо гра є потенційною, то пошук рівноваги Неша можна звести до задачі оптимізації P, що значно спрощує рішення. У нашій постановці функцію P можна визначити, наприклад, як сумарний показник SLA-виконання всіма сервісами.

Профіль стратегій $\sigma^* = (\sigma_1^*, \sigma_2^* \ldots, \sigma_n^*)$ називається рівновагою Неша, якщо для будь-якого сервісу i виконується нерівність:

$$\theta_i(\sigma_i^*, \sigma_{-i}^*) \geq \theta_i(\sigma_i, \sigma_{-i}^*) \forall \sigma_i \in \sum_i \quad (6)$$

Тобто жоден гравець не може односторонньо покращити свій виграш, змінюючи стратегію на іншу, якщо стратегії решти зафіксовані. У контексті ресурсів це означає: якщо всі інші сервіси зберігають свої частки ресурсів, жодному сервісу немає сенсу захопити більше ресурсів чи віддати частину, його власний показник θ_i від цього не покращиться. Рівновага Неша є природним кандидатом на стан системи, до якого вона може дійти шляхом самоорганізації сервісів.

У нашій грі рівновага може інтерпретуватися як стабільний розподіл ресурсів. На відміну від централізованого оптимального розподілу (коли, наприклад, ми максимізуємо $\sum_i \theta_i$ глобально), рівновага Неша досягається локально раціональними діями кожного сервісу. В ідеалі, якщо гра сконструйована вдало, така рівновага буде або співпадати з соціально-оптимальним станом, або принаймні знаходитися близько до нього за значенням потенційної функції. Відомо, що для потенційних ігор простий алгоритм послідовного поліпшення стратегій (де гравці по черзі переходять до стратегій, які покращують їхній виграш) гарантовано збігається до рівноваги Неша за скінченне число кроків [13].

На основі викладеної моделі можна запропонувати алгоритм, що наближує рівновагу Неша, дозволяючи мікросервісам поступово «домовитися» про розподіл ресурсів. В основі цього алгоритму лежить стратегічна гра між агентами.

Формальне позначення найкращої відповіді (best response) гравця (або мікросервісу) i на фіксовані стратегії інших гравців у певний момент часу відображено у наступному виразі:

$$\sigma_i^{best} := \arg\max_{\sigma_i \in \sum_i} \theta_i(\sigma_i, \sigma_{-i}^{t-1}) \quad (7)$$

σ_i^{best} - Найкраща стратегія для гравця i, arg max - аргумент, на якому досягається максимум, $\theta_i(.)$ - функція виграшу (корисності) гравця i, $\sigma_i \in \sum_i$ - множина всіх можливих стратегій гравця i, $\sigma_i, \sigma_{-1}^{t-1}$ - стратегії всіх інших гравців на попередній ітерації t−1.

Вхідними даними для алгоритму є: множина мікросервісів S, множина доступних ресурсів R, початковий розподіл $x^0 = x_{ir}^0$. Вихідними - рівноважний розподіл $x^* = x_{ir}^*$.

На початковому етапі встановлюється базовий розподіл ресурсів, наприклад, рівномірний, або з урахуванням попередньої статистики навантаження.

Далі система переходить до циклічного оновлення стратегій кожного сервісу. На кожній ітерації t кожен сервіс i ∈ S, знаючи поточний стан системи, визначає - чи може він покращити свій виграш, змінивши стратегію. Якщо може, сервіс переходить до нової стратегії. На основі цього обчислення сервіс обирає найкращу відповідь, згідно з виразом 7.

Нове значення розподілу встановлюється відповідно до обраної стратегії, тоді як стратегії інших сервісів залишаються незмінними, $x_{i*}^{(t)} := \sigma_i^{best}$, при $x_{j*}^{(t-1)} := x_{j*}^{(t-1)}$.

Процес повторюється, доки на черговій ітерації жоден із сервісів не знаходить для себе кращої стратегії, тобто система досягає рівноваги, $x^{(t)} = x^{(t-1)}$.

Алгоритм гарантовано завершується, оскільки кількість можливих станів системи (профілів стратегій) скінченна, а кожна ітерація покращує значення потенційної функції P або залишає його незмінним, а в потенційній грі не може бути нескінченного циклу покращень [13].

Звісно, наведений алгоритм - централізована (синхронна) версія (ми послідовно перебираємо сервіси). У реальній системі можна реалізувати його асинхронно: кожен сервіс (або агент, що діє від його імені) періодично перевіряє умови і за потреби коригує свій ресурсний запит. Самі обчислення оптимальної відповіді можуть виконуватись аналітично (якщо, скажімо, взяти похідну від θ_i і прирівняти до нуля, знайти оптимум) чи методом перебору (якщо простір стратегій дискретний і невеликий).

Для впровадження описаного підходу у середовище контейнерів розроблено компонент Game Theory Controller (GTC), що діє як надбудова до системи оркестрації (наприклад, Kubernetes). На рисунку 1 зображено схему архітектури з GTC. Контролер виконує роль «арбітра», який збирає інформацію про стан кожного мікросервісу (навантаження, поточний час відповіді, використання ресурсів) та обчислює новий розподіл ресурсів, керуючись ігровою моделлю.

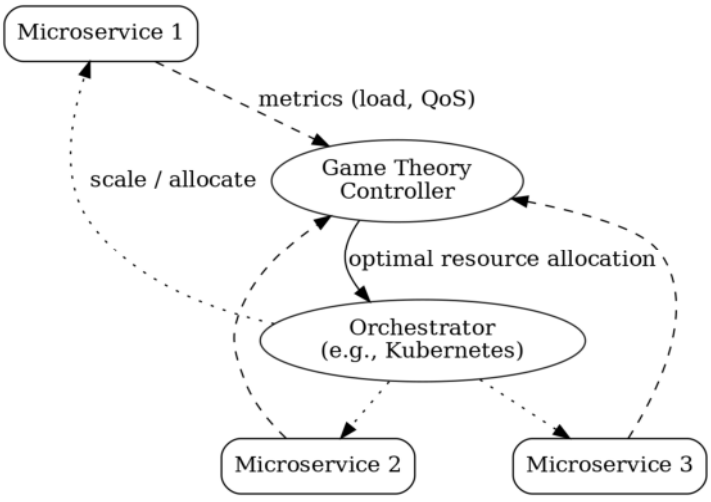

Рисунок 1 – Архітектура мікросервісної системи з контролером Game Theory Controller

Джерело: розроблено авторами

Як видно з рисунка 1, Game Theory Controller отримує метрики (показники навантаження, latency тощо) від моніторингової системи (яка може бути частиною платформи оркестрації або окремим агентом в кожному контейнері). На основі цих даних контролер розв’язує ігрову задачу: знаходить рівноважний розподіл ресурсів між

сервісами. Потім через API оркестратора контролер застосовує необхідні зміни в кластері.

Це може включати:

1. Масштабування окремих сервісів (збільшення або зменшення кількості реплік контейнера).

2. Міграцію контейнерів на інші вузли (якщо гра передбачає, що для рівноваги певний сервіс повинен працювати на іншому ресурсі).

3. Динамічне налаштування лімітів на ресурси (CPU, пам'ять) для контейнерів, що дозволяє контролювати, яку частку доступних ресурсів може використовувати кожен окремий сервіс.

Контролер працює у режимі замкнутого зворотного зв'язку: після кожної зміни він знов отримує оновлені метрики і перевіряє, чи система досягла стабільності. Якщо виявляється, що після перерозподілу якийсь сервіс почав отримувати гірший виграш (наприклад, через зміну характеру трафіку), процес продовжується - фактично, алгоритм 1 виконується у режимі реального часу, покроково наближаючи систему до рівноваги при поточному навантаженні.

У Kubernetes можна реалізувати GTC як кастомний контролер (operator), що взаємодіє з API Server. Наприклад, контролер може періодично збирати метрики з Prometheus (система моніторингу) і на їх основі розраховувати нові значення Replica Count для кожного Deployment (мікросервісу), або ж змінювати CPU requests/limits у Pod-ах. Розрахунок може виконуватися поза межами самого кластера (контролер як зовнішній сервіс) або всередині (у вигляді спеціального Pod-а).

Розглянемо спрощений сценарій: два мікросервіси (A і B) працюють на спільному кластері з двома вузлами. Сервіс A є критичнішим (його SLA жорсткіше, йому бажано низький час відповіді), сервіс B - менш чутливий. Припустимо, спочатку оркестратор розмістив обидва сервіси на одному вузлі. Внаслідок цього при високому навантаженні час відповіді A і B зріс, і A може порушувати SLA. Контролер отримує такі дані і «грає» за сервіс A, рекомендуючи йому стратегію, перенести деякі свої копії на другий вузол (або збільшити кількість копій, зайнявши другий вузол). Сервіс B, навпаки, може трохи зменшити свою частку CPU на першому вузлі або також запустити копію на другому. Після застосування, сервіс A отримав більше ресурсу ексклюзивно, його час відповіді повернувся в межі SLA, сервіс B теж працює прийнятно.

Жоден з них тепер не бажає змінювати розміщення - досягнуто рівноваги: A не хоче повертатися на спільний вузол з B, бо йому там гірше, а B не проти залишити A більше ресурсів, бо його власний показник при цьому все ще задовільний.

Така конфігурація відповідає інтуїтивному очікуванню: більш критичний сервіс отримав пріоритетний доступ до ресурсів.

Щоб оцінити вплив запропонованого підходу, корисно порівняти його результати з двома базовими політиками керування ресурсами, які часто застосовуються в оркестрації контейнерів: Binpack та Spread. Binpack прагне мінімізувати кількість зайнятих вузлів (щільно упаковуючи контейнери), а Spread - рівномірно розподілити навантаження між усіма вузлами. Game Theory Controller фактично намагається збалансувати інтереси сервісів з урахуванням поточного навантаження, що може нагадувати щось середнє між Binpack і Spread, але адаптивно.

У таблиці 1 підсумовано основні особливості цих стратегій.

Таблиця 1 – Порівняння стратегій розподілу ресурсів

Стратегія	Підхід	Переваги	Недоліки
Binpack	Максимально щільне розміщення сервісів на найменшій кількості вузлів.	Високе завантаження окремих вузлів, можливість вимкнути незалучені вузли для економії ресурсів.	Ризик перевантаження вузлів, погіршення продуктивності при піковому навантаженні; нерівномірний розподіл може призвести до гарячих точок.
Spread	Рівномірний розподіл сервісів по всіх доступних вузлах.	Уникає концентрації навантаження на одному вузлі, стабільний час відповіді при помірному навантаженні.	Недовикористання ресурсів: навіть при малому навантаженні залучені всі вузли (накладні витрати); може збільшувати міжвузлові затримки через розподіл сервісів.
Game Theory (з GTC)	Мікросервіси самостійно вибирають розподіл ресурсів, оптимальний для їх продуктивності (рівновага).	Адаптивний баланс ресурсів відповідно до актуальних потреб; поліпшення загальної продуктивності та справедливості (жоден сервіс не “обділений”).	Складність реалізації: потребує спеціального контролера та моделі; можливі коливання системи до встановлення рівноваги; необхідна достовірна інформація.

Джерело: розроблено авторами

Для більш наочного порівняння розглянемо сценарій зі зростаючим навантаженням на систему і виміряємо середній час відповіді сервісів при використанні різних політик. На рисинку 2 зображено залежність середнього часу відповіді від сумарного навантаження (умовно, кількості запитів за секунду) для стратегій Binpack, Spread та Game Theory (GTC, наш підхід). Криві отримано шляхом моделювання двох сервісів на двох вузлах: за малого навантаження всі стратегії дають низький час відповіді (~100 мс).

У міру зростання навантаження, Binpack починає стрімко погіршувати час відповіді (через перевантаження одного вузла), тоді як Spread і GTC зростають більш плавно. При дуже високому навантаженні (500 умовних одиниць) Binpack призводить до ~900 мс, Spread - ~350 мс, а Game Theory - ~280 мс. Таким чином, ігровий підхід наближує ефективність до оптимальної, перевершуючи обидві прості евристики.

З рисунка 2 видно, що при середніх і високих навантаженнях контролер на основі теорії ігор дозволяє підтримувати менший час відповіді, ніж Spread, уникнувши різкого росту затримки, як у випадку Binpack.

Це досягається завдяки динамічному перерозподілу: спочатку GTC може поводитися схоже на Binpack (за малого навантаження концентрувати сервіси, щоб решта ресурсів лишалася в запасі), але при наближенні до критичного завантаження він «розводить» сервіси по різних вузлах, як Spread. Водночас, якщо навантаження розподілено нерівномірно між сервісами, GTC надасть більш вимогливому сервісу більшу частку ресурсів, навіть якщо доведеться відібрати в інших, але настільки, наскільки це вигідно всім у довгостроковій перспективі (бо інакше інші теж постраждають від загального падіння продуктивності).

Оскільки система мікросервісів динамічна (навантаження змінюється постійно), Game Theory Controller повинен швидко реагувати та сходитися до нового балансу. Використання потенційної гри гарантує збіжність алгоритму найкращих відповідей, але в реальності можливі осциляції, якщо навантаження змінюється швидше, ніж встигає

спрацювати контролер. Для згладжування коливань можна впровадити інертність у прийнятті рішень (наприклад, не переміщувати сервіси занадто часто, чекати підтвердження тренду в метриках) [15].

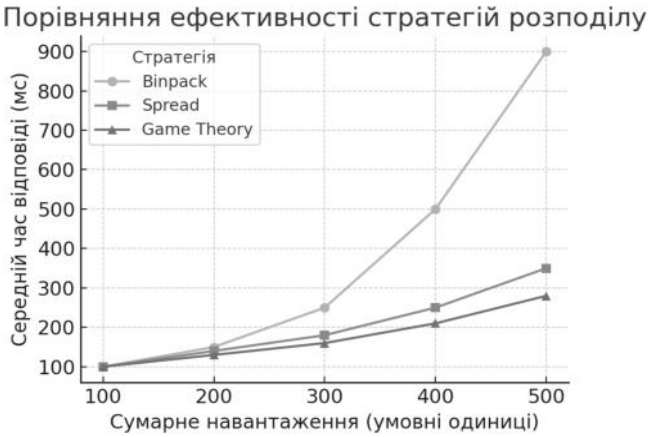

Рисунок 2 – Порівняння ефективності стратегій розподілу ресурсів (середній час відповіді залежно від навантаження)

Джерело: розроблено авторами

Обчислювальна складність контролера зростає з кількістю сервісів і вузлів. У найгіршому випадку, якщо кожен сервіс може бути на кожному з R ресурсів, простір стратегій експоненційний (R^n комбінацій). Однак на практиці n може бути обмеженим на окремому вузлі (оркестратор не розмістить надто багато подів на один вузол), і гра може розбиватися на менш розмірні під ігри (для кожного вузла окремо, або для кожного застосунку окремо, якщо в кластері кілька незалежних застосунків). Тому можливе модульне масштабування контролера, наприклад, запуск кількох екземплярів GTC, кожен з яких оптимізує свою підмножину сервісів.

У нашій реалізації приймається, що мікросервіси «грають чесно», тобто повідомляють контролеру правдиві метрики та слідують виданим ним рекомендаціям. Це справедливо, оскільки всі сервіси фактично належать одному власнику (розробнику застосунку) та діють на спільну мету - якісно обслугувати користувачів. На відміну від економічних ігор, тут немає мотиву обманювати або відхилятися від алгоритму, адже контролер прагне покращити їхні ж власні показники. Тому гру можна моделювати та розв'язувати централізовано без небезпеки маніпуляцій.

Запропонований підхід найкраще працює для сценаріїв, де взаємодія між сервісами відносно слабка, а основний фактор - конкуренція за спільні ресурси. Якщо ж мікросервіси мають сильні залежності (наприклад, послідовні виклики, що формують довгий ланцюжок), то оптимізація часу відповіді одного сервісу мало допоможе без оптимізації всього ланцюжка. В таких випадках доводиться розширювати модель, включаючи у виграш кожного сервісу і вплив затримки його сусідів. Це робить гру складнішою (функції виграшу переплітаються), але потенційний апарат все ще може бути застосований.

Висновки. У статті запропоновано підхід до оптимізації мікросервісної архітектури на основі теорії ігор. Виконано постановку проблеми розподілу обмежених ресурсів між множиною сервісів, що взаємодіють, як некооперативної гри та показано, що рівновага Неша в цій грі відповідає збалансованому розподілу ресурсів, який жоден

сервіс не зацікавлений змінювати одноосібно. На основі потенційної ігрової моделі розроблено алгоритм пошуку рівноваги та запропоновано архітектуру контролера Game Theory Controller для практичної реалізації такого підходу в системах оркестрації контейнерів.

Основні результати роботи включають:

1. Мікросервіси розглянуто як гравців, що конкурують за ресурси кластера, із формалізацією їхніх цілей через утиліти, пов'язані з показниками SLA/QoS. Показано, що проблему runtime-оптимізації можна звести до знаходження рівноваги Неша в цій грі.

2. Запропоновано ітеративний алгоритм керування ресурсами, який поступово наближається до рівноважного розподілу. Його реалізацію можна організувати як централізований контролер або у вигляді децентралізованої системи автономних агентів, вбудованих у сервіси.

3. Наведено дизайн контролера GTC, який інтегрується з існуючими засобами оркестрації (наприклад, Kubernetes) і здійснює моніторинг та переналаштування системи в автоматичному режимі. Архітектура протестована на спрощеній моделі і продемонструвала покращення продуктивності (зменшення часу відповіді, відсутність перевантажених вузлів) у порівнянні з типовими політиками.

4. Результати моделювання показують, що підхід на основі теорії ігор дозволяє знайти компроміс між полярними стратегіями, такими як Binpack і Spread. Система динамічно адаптується до поточного навантаження, уникаючи як простою ресурсів, так і їх перенавантаження, що забезпечує стабільну якість сервісів.

Водночас варто зауважити, що результати на цьому етапі дослідження є попередніми та грунтуються на моделюванні взаємодії лише між обмеженою кількостю сервісів. Теоретичне обґрунтування гарантій збіжності запропонованого методу, а також масштабування моделі на великі системи з великою кількістю агентів потребують подальшого вивчення.

Практичне значення отриманих результатів полягає в тому, що запропонований метод можна застосувати в реальних хмарних платформах для автономного керування мікросервісними застосунками. Він особливо корисний для систем з непостійним, важко передбачуваним навантаженням, де статичні або прості евристичні налаштування не справляються з підтриманням SLA. Теоретико-ігровий контролер доповнює наявні механізми автоскейлінгу, додаючи їм «свідомості» про присутність інших сервісів і глобальну картину. Планується розширити модель для врахування взаємозв'язків між сервісами (гра на графі викликів) - це приведе до багаторівневих ігор або ігор з коаліціями.

Таким чином, результати дослідження підтверджують, що теорія ігор має значний потенціал як інструмент для оптимізації мікросервісної архітектури. Водночас, для повноцінного застосування цього підходу потрібні подальші теоретичні дослідження та практичні експерименти. У цій роботі було розглянуто модель ітеративної взаємодії між обмеженою кількістю мікросервісів для ілюстрації методу. Щоб поширити його на великі системи, необхідно провести додатковий аналіз, який заплановано у наступних етапах дослідження.

Список літератури

1. Кулагін В. П., Улічев О. С., Доренський О. П. Інноваційні рішення та переваги мікросервісної архітектури програмних продуктів. *Центральноукраїнський науковий вісник. Технічні науки*. 2024. Вип. 10(41), ч. 1. С. 16–29. DOI: 10.32515/2664-262X.2024.10(41).16-29.

2. Dragoni N., et al. Microservices: How to make your application scale. *Lecture Notes in Computer Science*. 2018. Vol. 10742. P. 95–104. DOI: 10.1007/978-3-319-74313-4_8.
3. Altman E., et al. A survey on networking games in telecommunications. *Computers & Operations Research*. 2006. Vol. 33, № 2. P. 286–311. DOI: 10.1016/j.cor.2004.06.005.
4. Wei G., et al. A game-theoretic method of fair resource allocation for cloud computing services. *The Journal of Supercomputing*. 2010. Vol. 54, № 2. P. 252–269. DOI: 10.1007/s11227-009-0318-1.
5. Nash J. F. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*. 1950. Vol. 36, № 1. P. 48–49. DOI: 10.1073/pnas.36.1.48.
6. Ardagna D., Panicucci B., Passacantando M. Generalized Nash equilibria for the service provisioning problem in cloud systems. *IEEE Transactions on Services Computing*. 2013. Vol. 6, № 4. P. 429–442. DOI: 10.1109/TSC.2012.14.
7. Kaur K., et al. Container-as-a-Service at the Edge: Trade-off between Energy Efficiency and Service Availability at Fog Nano Data Centers. *IEEE Wireless Communications*. 2017. Vol. 24, № 3. P. 48–56. DOI: 10.1109/MWC.2017.1600427.
8. Yan S., et al. An Evolutionary Game for User Access Mode Selection in Fog Radio Access Networks. *IEEE Access*. 2017. Vol. 5. P. 2200–2210. DOI: 10.1109/ACCESS.2017.2654266.
9. Luo R., et al. Runtime Resource Management for Microservices-Based Applications: A Congestion Game Approach. *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering (LNICST)*. 2019. Vol. 268. P. 676–687. DOI: 10.1007/978-3-030-12981-1_47.
10. He Q., Wang H., Jin H. A Game-Theoretical Approach for User Allocation in Edge Computing Environment. *IEEE Transactions on Parallel and Distributed Systems*. 2020. Vol. 31, № 4. P. 515–529. DOI: 10.1109/TPDS.2019.2938944.
11. Kumar S., et al. A Game-Theoretic Approach for Increasing Resource Utilization in Edge Computing Enabled IoT. *IEEE Access*. 2022. Vol. 10. P. 57974–57989. DOI: 10.1109/ACCESS.2022.3175850.
12. Na J., et al. An evolutionary game approach on IoT service selection for balancing device energy consumption. *Proc. 12th IEEE Int. Conf. on e-Business Engineering (ICEBE)*. 2015. P. 331–338. DOI: 10.1109/ICEBE.2015.64.
13. Monderer D., Shapley L. S. Potential Games. *Games and Economic Behavior*. 1996. Vol. 14, № 1. P. 124–143. DOI: 10.1006/game.1996.0044.
14. Velasquez K., et al. Service placement for latency reduction in the Internet of Things. *Annals of Telecommunications*. 2017. Vol. 72, № 1–2. P. 105–115. DOI: 10.1007/s12243-016-0524-9.
15. Baresi L., et al. A discrete-time feedback controller for containerized cloud applications. *Proc. ACM SIGSOFT Int. Symp. Foundations of Software Engineering (FSE)*. 2016. P. 217–228. DOI: 10.1145/2950290.2950328.
16. Guerrero C., Lera I., Juiz C. Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications. *The Journal of Supercomputing*. 2018. Vol. 74, № 7. P. 2956–2983. DOI: 10.1007/s11227-018-2345-2.

References

1. Kulahin, V. P., Ulichev, O. S., & Dorenskyi, O. P. (2024). Innovative Solutions and Benefits of Microservice Architecture for Software Products. *Central Ukrainian Scientific Bulletin. Technical Sciences, 10(41, Part 1)*, 16–29. https://doi.org/10.32515/2664-262X.2024.10(41).16-29.
2. Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R., & Safina, L. (2018). Microservices: How to make your application scale. *Lecture Notes in Computer Science*, *10742*, 95–104. https://doi.org/10.1007/978-3-319-74313-4_8.
3. Altman, E., Boulogne, T., El-Azouzi, R., Jiménez, T., & Wynter, L. (2006). A survey on networking games in telecommunications. *Computers & Operations Research*, *33(2)*, 286–311. https://doi.org/10.1016/j.cor.2004.06.005.
4. Wei, G., Vasilakos, A. V., Zheng, Y., & Xiong, N. (2010). A game-theoretic method of fair resource allocation for cloud computing services. *The Journal of Supercomputing*, *54(2)*, 252–269. https://doi.org/10.1007/s11227-009-0318-1.
5. Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences, 36(1),* 48–49. https://doi.org/10.1073/pnas.36.1.48.
6. Ardagna, D., Panicucci, B., & Passacantando, M. (2013). Generalized Nash equilibria for the service provisioning problem in cloud systems. *IEEE Transactions on Services Computing*, *6(4)*, 429–442. https://doi.org/10.1109/TSC.2012.14.

7. Kaur, K., Dhand, T., Kumar, N., & Zeadally, S. (2017). Container-as-a-Service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers. *IEEE Wireless Communications*, *24(3)*, 48–56. https://doi.org/10.1109/MWC.2017.1600427.
8. Yan, S., Peng, M., Abana, M. A., & Wang, W. (2017). An evolutionary game for user access mode selection in fog radio access networks. *IEEE Access, 5,* 2200–2210. https://doi.org/10.1109/ACCESS.2017.2654266.
9. Luo, R., Ye, W., Sun, J., Liu, X., & Zhang, S. (2019). Runtime resource management for microservices-based applications: A congestion game approach. In *Proceedings of the 14th International Conference on Collaborative Computing (CollaborateCom), LNICST 268* (pp. 676–687). https://doi.org/10.1007/978-3-030-12981-1_47.
10. He, Q., Wang, H., Jin, H., et al. (2020). A game-theoretical approach for user allocation in edge computing environment. *IEEE Transactions on Parallel and Distributed Systems, 31(4),* 515–529. https://doi.org/10.1109/TPDS.2019.2938944.
11. Kumar, S., Sharma, V., You, I., et al. (2022). A game-theoretic approach for increasing resource utilization in edge computing enabled IoT. *IEEE Access, 10,* 57974–57989. doi.org/10.1109/ACCESS.2022.3175850.
12. Na, J., Lin, K.-J., Huang, Z., & Zhou, S. (2015). An evolutionary game approach on IoT service selection for balancing device energy consumption. In *Proceedings of the 12th IEEE International Conference on e-Business Engineering (ICEBE)* (pp. 331–338). https://doi.org/10.1109/ICEBE.2015.64.
13. Monderer, D., & Shapley, L. S. (1996). Potential games. *Games and Economic Behavior, 14(1),* 124–143. https://doi.org/10.1006/game.1996.0044.
14. Velasquez, K., Abreu, D. P., Curado, M., & Monteiro, E. (2017). Service placement for latency reduction in the Internet of Things. *Annals of Telecommunications, 72(1–2),* 105–115. https://doi.org/10.1007/s12243-016-0524-9
15. Baresi, L., Guinea, S., Leva, A., & Quattrocchi, G. (2016). A discrete-time feedback controller for containerized cloud applications. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)* (pp. 217–228). https://doi.org/10.1145/2950290.2950328.
16. Guerrero, C., Lera, I., & Juiz, C. (2018). Resource optimization of container orchestration: A case study in multi-cloud microservices-based applications. *The Journal of Supercomputing, 74(7),* 2956–2983. https://doi.org/10.1007/s11227-018-2345-2.

Oleksandr Ulichev[1], PhD tech. sci, **Victor Kulahin**[2]
[1]*Central Ukrainian National Technical University, Kropyvnytskyi, Ukraine*
[2]*Private Higher Education Establishment "European University", Kyiv, Ukraine*

Game-Theoretic Approach to Microservice Optimization

Modern software systems increasingly adopt microservice architectures (MSA) to achieve modularity, scalability, and independent deployment. However, the decentralized nature of microservices introduces complex challenges in resource allocation, load balancing, and maintaining system-wide performance under dynamic workloads. Traditional orchestration methods often rely on heuristic or static rules that are insufficient for optimizing resource usage in highly variable and interactive environments.

This research explores the application of game theory as a formal framework for modeling and optimizing interactions among microservices. In this approach, each microservice is treated as a rational agent or player that independently selects strategies for resource consumption, scaling, or request routing. By applying models of non-cooperative games, such as congestion games, we identify equilibrium states (e.g., Nash equilibrium) that ensure stable and fair allocation of limited resources. In cooperative settings, game-theoretic mechanisms like Nash Bargaining can promote system-wide optimization through strategic coordination.

Simulation results demonstrate that game-theoretic strategies can significantly improve performance metrics, including average response time, resource utilization, and system resilience, compared to conventional approaches. Moreover, the integration of game-theoretic models with machine learning enables adaptive decision-making, allowing services to update strategies based on observed system states and predicted loads.

The paper shows that game theory provides a powerful and scalable foundation for the self-optimization of microservice-based systems. It opens new possibilities for designing intelligent orchestration layers capable of dynamically balancing autonomy and coordination in distributed software environments.

microservice architecture, optimization, game theory, Nash equilibrium, resource allocation, container orchestration, quality of service (QoS)

Одержано (Received) 16.06.2025 *Прорецензовано (Reviewed) 25.07.2025*
Прийнято до друку (Approved) 27.08.2025