https://doi.org/10.31891/2307-5732-2026-363-46
УДК 681.5:004.89

ЛЯШЕНКО ОЛЕКСІЙ
Харківський Національний Університет Радіоелектроніки
https://orcid.org/0000-0002-0146-3934
e-mail: oleksii.liashenko@nure.ua
МИХАЙЛІЧЕНКО ІГОР
Харківський Національний Університет Радіоелектроніки
https://orcid.org/0009-0005-5476-8992
e-mail: ihor.mykhailichenko@nure.ua

МОДЕЛЬ САМОАДАПТИВНОЇ РОЗПОДІЛЕНОЇ СИСТЕМИ КЕРУВАННЯ РЕСУРСАМИ У ХМАРНИХ ОБЧИСЛЕННЯХ

У роботі представлено децентралізований підхід до самоадаптивного керування ресурсами хмарної інфраструктури на основі мережі взаємодіючих агентів із локальними прогнозними моделями та протоколом консенсусу. Запропонована архітектура поєднує механізми прогнозування навантаження, локального аналізу телеметрії та P2P-синхронізації станів, формуючи розподілений контур MAPE (Monitor – Analyze – Plan – Execute), який працює без центрального контролера. На рівні кожного вузла реалізовано автономний агент із можливістю колективного прийняття рішень щодо масштабування сервісів у реальному часі. Для оцінювання ефективності розроблено симуляційне середовище, яке відтворює складні навантажувальні профілі та аварійні стани кластерів Kubernetes. Проведено порівняння децентралізованої архітектури з класичним централізованим автоскейлером. Експериментальні результати показали, що запропонована P2P-система забезпечує істотно швидший відгук на раптові зміни навантаження, меншу кількість перевантажень CPU, скорочений час відновлення після відмов вузлів і стабільність масштабування навіть під час мережевих розділень.

Ключові слова: самоадаптивні системи, хмарні обчислення, децентралізована архітектура, P2P-взаємодія, нейронні мережі для прогнозування навантаження, консенсусні алгоритми.

LYASHENKO OLEKSII, MYKHAYLICHENKO IHOR
Kharkiv National University of Radio Electronics

MODEL OF A SELF-ADAPTIVE DISTRIBUTED RESOURCE MANAGEMENT SYSTEM IN CLOUD COMPUTING

This paper presents a decentralized approach to self-adaptive resource management in cloud infrastructures based on a network of interacting agents equipped with local predictive models and a consensus protocol. The proposed architecture integrates workload forecasting, local telemetry analysis, and P2P state synchronization mechanisms, forming a distributed MAPE (Monitor–Analyze–Plan–Execute) control loop that operates without a central controller. At the level of each node, an autonomous agent is implemented with the capability of collective decision-making for real-time service scaling. Unlike traditional centralized autoscaling mechanisms, the proposed system distributes control logic across cluster nodes, enabling local decision-making and coordinated global behavior through consensus. Each agent continuously analyzes multi-dimensional telemetry data, predicts workload trends using neural network-based models, and exchanges state information with neighboring agents. Consistency of scaling decisions is ensured by a consensus protocol while preserving fault tolerance under partial failures.

To evaluate the effectiveness of the proposed approach, a simulation environment was developed that reproduces complex workload patterns and failure scenarios of Kubernetes clusters, including load spikes, node failures, and network partitions. A comparative analysis between the decentralized architecture and a classical centralized autoscaling mechanism was conducted under identical conditions. Experimental results demonstrate that the proposed P2P-based system provides a significantly faster response to sudden workload changes, reduces CPU overload events, shortens recovery time after node failures, and maintains stable scaling behavior during network partitions. Although the P2P architecture introduces additional communication overhead, its impact remains negligible relative to overall cluster traffic. The results confirm that decentralized self-adaptive resource management is an effective alternative to centralized autoscaling in dynamic cloud-native environments.

***Keywords**: self-adaptive systems, cloud computing, decentralized architecture; P2P interaction; neural networks for workload forecasting; consensus algorithms.*

Стаття надійшла до редакції / Received 02.01.2026
Прийнята до друку / Accepted 11.02.2026
Опубліковано / Published 26.03.2026

Постановка проблеми

Сучасні хмарні обчислювальні системи, побудовані за принципами cloud-native архітектури, значною мірою покладаються на централізовані механізми управління ресурсами, реалізовані оркестраторами на кшталт Kubernetes. Компоненти центрального контролера – Scheduler, Controller Manager та Autoscaler – формують глобальну точку прийняття рішень щодо планування контейнерів, горизонтального масштабування та розподілу навантаження. Така архітектура ефективна в умовах стабільної мережевої топології, проте створює низку системних обмежень: залежність від централізованого вузла управління, знижену відмовостійкість при втраті з'єднання з API-сервером, високі комунікаційні витрати на передачу телеметрії, а також затримки в реакції на локальні перевантаження. У сценаріях edge/fog-обчислень, розподілених GPU-кластерів, мульти-регіональних або частково ізольованих інфраструктур, ці недоліки стають критичними.

У роботі вперше запропоновано модель самоадаптивної розподіленої системи керування ресурсами, у

якій функції моніторингу, прогнозування та прийняття рішень повністю децентралізовано між вузлами кластера. На кожному вузлі розгортається локальний агент з вбудованим MAPE-контуром (Monitor–Analyze–Plan–Execute), що виконує збір апаратних та системних метрик (CPU utilization, memory pressure, disk I/O saturation, network RTT/throughput), здійснює коротко- та середньострокове прогнозування навантаження за допомогою вбудованої моделі та приймає рішення про локальне масштабування контейнерів або перерозподіл робочих процесів. Агенти утворюють P2P-мережу з прямим обміном телеметрією, можливостями побудови накладної топології (overlay), виявлення сусідів та формування узгоджених рішень без участі центрального планувальника.

Ключовим аспектом є те, що глобальні керувальні дії, такі як: створення нових екземплярів сервісу, переміщення подів між вузлами, обмеження або розширення ресурсних квот – приймаються колективно на основі розподіленого алгоритму консенсусу. Використання fault-tolerant протоколів Raft або їхніх легковагових модифікацій забезпечує гарантовану узгодженість стану, реплікацію керувальних сигналів між вузлами, толерантність до відмов окремих агентів і коректне функціонування при часткових збоях мережі. Такий підхід дозволяє реалізувати розподілений аналог класичного Scheduler'a Kubernetes, де роль лідера може динамічно переходити між вузлами, а рішення приймаються децентралізовано.

Запропонована модель «peer-to-peer autoscaling» усуває єдину точку відмови, знижує затримки прийняття рішень за рахунок локальної телеметрії, скорочує мережеві накладні витрати та забезпечує масштабованість у середовищах зі змінною топологією. На відміну від централізованих систем, локальні агенти мають безпосередній доступ до апаратних лічильників, можуть керувати контейнерами через Docker/Kubernetes API або напряму через CRI, а також здатні функціонувати автономно у випадку недоступності контролера.

Аналіз досліджень та публікацій

Останні роки характеризуються інтенсивним розвитком методів інтелектуального аналізу телеметрії та автоматизованого керування ресурсами у хмарних обчислювальних системах. Особлива увага приділяється застосуванню глибоких нейронних мереж, зокрема трансформерних архітектур, для прогнозування навантаження, виявлення аномалій і підтримки процесів прийняття рішень у складних розподілених середовищах.

У роботі Song et al. [2] досліджено застосування трансформерних моделей для прогнозування багатовимірних часових рядів телеметрії в системах керування життєзабезпеченням космічних апаратів. Автори демонструють високу точність довгострокового прогнозування за рахунок механізму self-attention, однак розглянутий підхід орієнтований на централізовану обробку даних і не враховує обмеження, характерні для хмарних систем реального часу, зокрема затримки керування та децентралізований характер інфраструктури.

У статті Smendowski et al. [3] запропоновано підхід до оптимізації використання ресурсів хмарних платформ шляхом багатогоризонтного прогнозування часових рядів. Показано, що поєднання кількох часових масштабів дозволяє зменшити помилки прогнозу та підвищити ефективність автоскейлінгу. Водночас запропонована модель не розглядає проблеми відмовостійкості та залежить від централізованого механізму прийняття рішень.

Дослідження Liu et al. [4] присвячене схемам ефективного розподілу ресурсів у середовищі Kubernetes для workflow-орієнтованих обчислень. Автори пропонують механізми динамічного виділення ресурсів на основі поточних метрик, однак аналіз виконується переважно на рівні окремих робочих процесів і не охоплює складні сценарії деградацій, мережевих розділень та масштабних пікових навантажень.

У роботі Taheri et al. [5] розглянуто використання методів машинного навчання для точного прогнозування споживання ресурсів Kubernetes-кластерів. Запропоновані моделі демонструють хорошу точність, проте вони застосовуються виключно як аналітичний компонент і не інтегровані безпосередньо в контур автономного керування ресурсами.

Значна кількість робіт присвячена виявленню аномалій у хмарних середовищах. Зокрема, Lian et al. [6] пропонують багатошкальний підхід до моделювання часових рядів для детекції аномальних станів сервісів, що дозволяє враховувати як короткочасні сплески, так і довгострокові тренди. Подібну проблематику розглядає Yu [7], де представлено модель DTAAD, що поєднує TCN та механізми уваги для аналізу багатовимірних часових рядів. Водночас обидва підходи орієнтовані переважно на офлайн-аналіз і не вирішують задачу узгодженого прийняття керувальних рішень у розподіленому середовищі.

У роботі Chakraborty та Heintz [8] досліджено модифікацію трансформерних моделей із використанням нечіткої уваги, що підвищує стійкість прогнозування до шуму. TranAD [9], запропонована Tuli et al., демонструє високу точність у задачах виявлення аномалій у багатовимірних часових рядах, однак потребує значних обчислювальних ресурсів і не адаптована для використання у вузлових агентних компонентах.

Таким чином, аналіз сучасних публікацій свідчить, що більшість існуючих підходів зосереджені або на централізованому прогнозуванні навантаження, або на локальній оптимізації використання ресурсів без урахування проблем узгодженості, відмовостійкості та автономності керування. Невирішеною залишається задача побудови самоадаптивної децентралізованої системи керування ресурсами, яка поєднує інтелектуальні прогнозні моделі з P2P-взаємодією агентів і консенсусним прийняттям рішень. Запропоноване в цій роботі рішення спрямоване на усунення зазначених обмежень і розвиток існуючих підходів у напрямі повноцінних автономних cloud-native систем керування.

Формулювання цілей статті

Метою дослідження є розробка, формалізація та експериментальна оцінка моделі самоадаптивної розподіленої системи керування ресурсами у хмарних обчисленнях, у якій функції моніторингу, прогнозування, планування та виконання управляючих дій децентралізовано між вузлами кластера, а узгодженість рішень забезпечується P2P-координацією та fault-tolerant механізмом консенсусу.

Модель повинна забезпечувати можливість локального та глобального масштабування, толерантність до відмов частини вузлів, роботу в умовах часткової або повної недоступності центрального контролера та інтеграцію у симуляційне cloud-native середовище.

Для досягнення мети необхідно вирішити такі технічні задачі дослідження:

– Визначити архітектурні та функціональні вимоги до агентів, P2P-протоколів та механізмів консенсусу, а саме: перелік телеметричних метрик, які обов'язково повинні бути локально доступними, допустимі межі латентності прийняття рішень, обмеження на мережеві накладні витрати при широкомовних або групових P2P-обмінах;

– Проєктування архітектури локального агента з MAPE-циклом;

– Проєктування протоколу обміну станами між агентами з урахуванням механізмів виявлення сусідів (node discovery), оновлення топології та побудови overlay-мережі, моделей періодичної синхронізації (gossip, epidemic broadcast) або запит-орієнтованих обмінів, формату повідомлень про локальні запити масштабування, конфлікти ресурсів, зведені метрики;

– Розробка та формалізування алгоритма узгодження масштабування на основі Raft або lightweight-модифікації;

– Реалізація експериментального середовища, яке дозволяє емулювати поведінку вузлів і контейнерів, запуск процесів та генерацію навантаження, моделювати пікові навантаження, відмови вузлів, коливання мережевої затримки.

Виклад основного матеріалу

Структура системи

Розроблена модель самоадаптивної розподіленої системи керування ресурсами побудована за принципом однорідних вузлів, кожен з яких містить локального агента з MAPE-контуром та бере участь у P2P-мережі консенсусу [1]. Відсутній постійний централізований контролер: глобальні рішення формуються колективно, а всі керувальні дії застосовуються до кластера через стандартні інтерфейси Kubernetes. Архітектура самоадаптивної системи, показана на рис. 1.

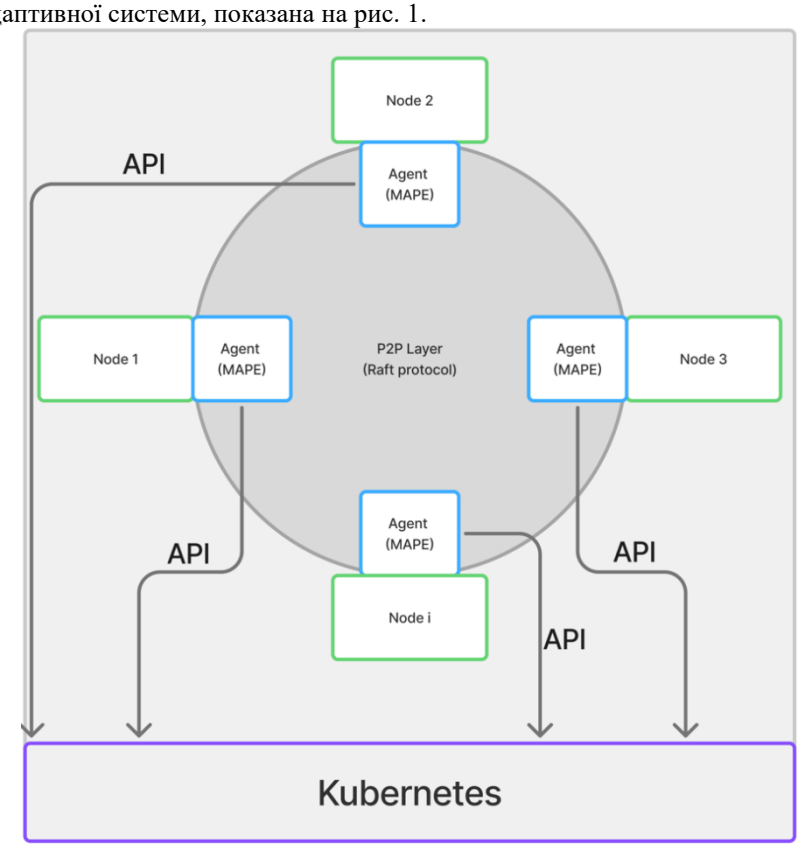

Рис.1. Загальна схема самоадаптивної розподіленої системи

На логічному рівні система складається з таких компонентів:

– Вузол кластера Node_i – фізичний або віртуальний сервер, на якому працюють pod'и, контейнерний runtime та локальний агент;

– Локальний агент Agent_i – процес/контейнер, що реалізує MAPE-контур;

– P2P-комунікаційний шар – накладна мережа між агентами для обміну станами, ініціації масштабування та проведення раундів консенсусу;

– Підсистема консенсусу – реалізація протоколу Raft для досягнення узгоджених рішень;

– API-взаємодія з Kubernetes – набір викликів (kube-apiserver / CRI / kubectl wrapper), через які агенти фактично змінюють стан кластера.

Архітектура моделі

Запропонована в роботі модель реалізує самоадаптивну розподілену архітектуру управління ресурсами, в якій кожен вузол кластера Kubernetes виконує роль автономного інтелектуального агента. Архітектура моделі передбачає відмову від централізованого контролера та перехід до децентралізованої координації, що забезпечується комбінацією P2P-взаємодії та протоколу консенсусу типу Raft.

Кожен вузол Node_i містить модуль Agent_i, який реалізує локальний цикл керування Monitor–Analyze–Plan–Execute. Агент працює як окремий pod або systemd-сервіс та має доступ до локальних метрик ядра й Kubernetes-контролерів [2].

Архітектура Agent_i включає такі підсистеми (рис. 2):

– Модуль моніторингу реалізовано як багатопотоковий компонент, що взаємодіє з такими джерелами телеметрії: cAdvisor (CPU/memory/IO/Network подів), kubelet metrics endpoint(статуси контейнерів, throttling), node-exporter/OS-інтерфейси (системні метрики вузла), Kubernetes API (кількість реплік, статуси сервісів);

– Модуль Analyze, який виконує локальний прогноз навантаження та детекцію ризикових станів. Його архітектура включає: Feature Extractor, прогностичний блок, блок виявлення ризиків, класифікатор сценаріїв;

– Модуль Plan, що відповідає за формування локальної або глобальної дії;

– Модуль Execute є інкапсуляцією Kubernetes API та підтримує такі операції: масштабування Deployment/StatefulSet, створення нових pod'ів на конкретному вузлі.

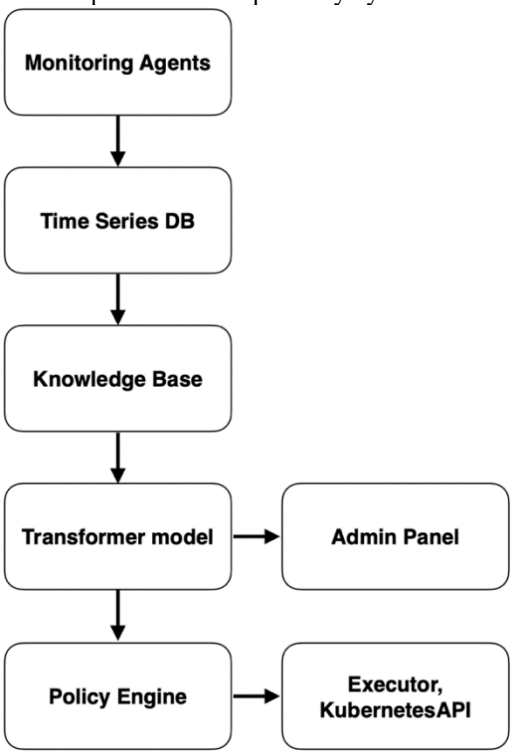

Рис.2. Загальна архітектура Agent_i

P2P-комунікаційний шар забезпечує горизонтальну координацію агентів. Будується однорангова мережа з частковою топологією: full-mesh при $N \leq 20$, hybrid mesh-ring при $20 < N < 200$, Kademlia-DHT при $N2 \geq 200$. Кожне з'єднання підтримується через канали gRPC із двома потоками: Control Stream (для консенсусу) та Telemetry Stream (для обміну агрегованими станами) [3].

Повідомлення будуються як Protobuf-структур. Підтримуються три класи повідомлень: управлінські (CONTROL – consensus propose/vote/commit), оглядові (TELEMETRY – передача агрегованих станів), системні (SYSTEM – heartbeat, ping, reconnect).

Система консенсусу забезпечує гарантоване узгодження масштабування між вузлами. На кожному агенті підтримується журнал $Log_i = \{entry_1, entry_2, \dots, entry_k\}$, де $entry = (term, index, cmd)\}$. Журнал зберігається в локальному SQLite для збереження між перезапусками.

Важливою складовою є протокол консенсусу (Raft-like). Протокол має такі фази:

– Election – вузли з тайм-аутом стають кандидатами, надсилають запити vote-request, коли отримують більшість, стають лідером;

– Replication – лідер додає нову команду cmd у свій журнал, розсилає її як AppendEntries;

– Commit – при отриманні кворуму підтверджень, команда стає commited. Після чого лідер розсилає ACTION_COMMIT повідомлення;

– Apply – модуль Execute на вузлі-цілі виконує команду.

Наступним починає працювати модуль планування та виконання Kubernetes. Після затвердження команди консенсусом, зміна стану кластера виконується через Execute-шар.

Запропонована система використовує спеціалізований P2P-комунікаційний шар, який забезпечує децентралізовану координацію агентів без використання централізованих контролерів. Архітектура розроблена таким чином, щоб витримувати реалістичні мережеві затримки, втрати пакетів, тимчасові розриви з'єднань та сценарії часткових відмов вузлів. P2P-шар є ключовим елементом системи, оскільки саме через нього реалізується розподілений консенсус, обмін локальними станами вузлів, передача запитів глобального масштабування та синхронізація журналів [4].

Усі агенти формують повністю децентралізовану однорангову мережу, побудовану поверх реальної мережевої інфраструктури кластера. З точки зору логіки протоколів, мережа відтворює властивості: асинхронності (немає узгодженого глобального часу), недетермінованої доставки повідомлень, можливості мережевих поділів (split-brain), відсутності гарантії порядку доставки, можливості втрати або дублювання повідомлень. На кожному каналі між Agent_i та Agent_j підтримуються два окремі внутрішні стріми.

Control Stream використовуються для повідомлень про консенсус, запитів глобального масштабу, сповіщення про фіксацію. Цей стрім має підвищений пріоритет, внутрішній QoS та менший розмір буферів для мінімізації затримок.

Telemetry Stream використовуєтсья для періодичного обміну агрегованими станами вузлів, передачі метрик та оновлення прогнозів. Таке розділення гарантує, що консенсус та глобальні рішення не блокуються при високому трафіку телеметрії.

Постановка експерименту

Для експериментальної верифікації запропонованої самоадаптивної розподіленої моделі керування ресурсами було розроблено симуляційне середовище, у якому здійснюється порівняння двох принципово різних підходів до масштабування хмарних інфраструктур: централізованого, реалізованого засобами Kubernetes Horizontal Pod Autoscaler (HPA) та Cluster Autoscaler (CA) [5], і децентралізованого, побудованого на запропонованій P2P-архітектурі локальних агентів з механізмами консенсусу. Обидва підходи аналізуються в ідентичних умовах, що забезпечує коректність порівняння та ізоляцію впливу архітектурних відмінностей на поведінку системи.

Експериментальне середовище моделює Kubernetes-кластер із двадцяти однотипних вузлів, кожен з яких має конфігурацію 4 CPU та 8 GB оперативної пам'яті. Мережеві затримки між вузлами визначаються в діапазоні 5-15 мс та доповнюються стохастичним джиттером, що адекватно відтворює умови взаємодії між вузлами у типовому overlay CNI середовищі (Calico). Оркестратор Kubernetes версії 1.29 працює з контейнерним рантаймом containerd та типовим kube-scheduler. Усі експерименти проводяться для одного сервісу, який обробляє веб-запити та має pod із конфігурацією 0.25 CPU та 128 MB memory request. Для нього характерна висока чутливість до миттєвих пікових навантажень та throttling-поведінки ядра Linux у разі браку CPU [6].

Робоче навантаження формується синтетичною моделлю, яка імітує поведінку веб-сервісу з добовою сезонністю, випадковими шумами та короткочасними піковими burst-навантаженнями. Інтенсивність запитів описується функцією $\lambda(t) = \lambda_0 + A\sin(\omega t + \phi) + B\mathit{burst}(t) + \eta(t)$, де базовий рівень $\lambda_0 = 180$ RPS доповнюється синусоїдальною складовою з амплітудою 90 та періодом в одну годину, імпульсними навантаженнями з амплітудою до п'ятикратного перевищення бази та шумовою компонентою. Такий підхід дозволяє генерувати реалістичні сценарії коливань навантаження, у тому числі різкі піки, що становлять ключовий виклик для систем масштабування. Для аналізу стійкості системи вводяться також два типи аномалій: CPU-spike (раптове збільшення навантаження у 5 разів на короткому інтервалі), а також memory-leak pod, який поступово збільшує використання пам'яті.

У централізованій схемі baseline використовується HPA, який регулює кількість pod'ів на основі метрики CPU utilization із цільовим значенням 70%. HPA працює у межах від 2 до 16 реплік та володіє типовим стабілізаційним вікном тривалістю 300 секунд, що значно знижує швидкість реакції при різких коливаннях навантаження [7]. Cluster Autoscaler відповідає за масштабування інфраструктури на рівні вузлів: він дозволяє динамічно збільшувати кількість вузлів у межах від 10 до 30 та застосовує доволі інерційні політики scale-down, включно з очікуванням 10 хвилин після масштабування для ухвалення рішень про зменшення кластеру.

Для узгодження рішень агентів застосовується модифікована версія протоколу Raft, у якій «election timeout» визначається рівномірним розподілом у діапазоні 150-300 мс, що забезпечує низьку латентність виборів та мінімізує конфлікти кандидатур. Кворум визначається як більшість вузлів (11 із 20) [8]. Журнал команд масштабування реплікується як ланцюг хешів SHA-256, що гарантує незмінність та виявлення розходжень у разі network split. У випадку втрати зв'язку між частиною вузлів, агенти без кворуму переходять у read-only режим, натомість інша частина мережі продовжує працювати та ухвалювати рішення без втрати узгодженості.

Експериментальні дослідження охоплюють декілька сценаріїв. У першому оцінюється здатність систем реагувати на сезонне навантаження з імпульсними піками, що дозволяє визначити час реакції та кількість SLA-порушень у моменти пікового навантаження. У другому аналізується поведінка системи під час

CPU-spike. Ключовими показниками є ступінь перевантаження вузлів та час виходу у стабільний режим. У третьому досліджується memory-leak сценарій, що дозволяє оцінити автономність та швидкість локальних дій агентів у порівнянні з централізованим HPA [9]. У четвертому виконується моделювання часткового мережевого поділу, що дозволяє оцінити стійкість консенсусу та поведінку журналів після відновлення зв'язку. У п'ятому створюється сценарій раптової відмови вузлів для аналізу fault-tolerance обох систем.

Оцінювання ефективності здійснюється за кількома показниками, серед яких час ухвалення масштабувальних рішень, кількість SLA-порушень, рівень перевантаження вузлів, кількість надлишкових масштабувань, латентність commit-операцій у консенсусі та стабільність журналу. Така комплексна методика дозволяє дослідити не лише швидкість реагування, але й системну поведінку кожного підходу в умовах нестабільності та часткових відмов.

Результати дослідження

На рис. 3 наведено часові ряди кількості реплік pod'ів для двох підходів автоскейлінгу протягом шестигодинного експерименту: централізованого (HPA + Cluster Autoscaler) та запропонованої децентралізованої P2P-системи локальних агентів. Вісь абсцис відповідає часу (у годинах), вісь ординат – кількості активних реплік сервісу.

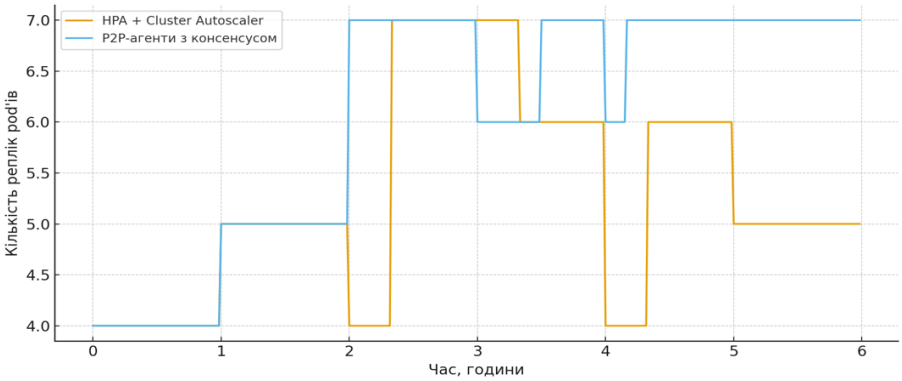

Рис.3. Порівняння динамік автоскейлінгу

У першій фазі експерименту (до 2 годин), моделюється плавне сезонне зростання навантаження. Обидві системи поводяться подібно: кількість реплік зростає з 4 до 5 pod'ів, а різниця між HPA та P2P є мінімальною. Це показує, що за повільних, добре передбачуваних змін трафіку класичний HPA залишається цілком достатнім і конкурентоспроможним.

Суттєві відмінності проявляються під час раптового «CPU-spike» на межі другої години. У момент різкого збільшення інтенсивності запитів P2P-система реагує майже миттєво: на рис.3 видно, що вже в інтервалі 2.0-2.5 год кількість реплік у P2P зростає до 7. Це зумовлено використанням локального прогнозу навантаження та P2P-синхронізації станів, які дозволяють прийняти колективне рішення про масштабування до входження системи у режим насичення [10]. Централізований HPA, навпаки, демонструє типовий запізнілий відгук: вихід на 7 реплік відбувається лише через 20-30 хв після піку, коли крива HPA+CA досягає відповідного рівня. У цей період система працює на піку, що призводить до зростання порушень SLA та пікового завантаження CPU.

Після спаду навантаження (близько 3-ї години) HPA демонструє гістерезис: певний час утримує завищену кількість реплік і далі зменшує її з характерними коливаннями між 5 та 6 pod'ами. P2P-система формує більш плавну траєкторію: після піку вона знижує масштабування до 6 реплік, але не виконує агресивного scale-down до мінімуму, залишаючи невеликий запас потужності. Відповідно, коливання масштабування у P2P-випадку менш виражені, що вказує на вищу стабільність алгоритму за рахунок колективного погодження рішень.

Друга група подій включає витік пам'яті, відмову вузла та мережеве розділення. На інтервалі близько 3.5 години моделюється сценарій memory-leak в одному з pod'ів. Для централізованого варіанта, орієнтованого лише на CPU, цей ефект практично не впливає на рішення HPA, і крива HPA+CA залишається на рівні 6 реплік. У P2P-системі превентивна логіка агента фіксує аномальний тренд використання пам'яті, тому крива раніше виходить на 7 pod'ів – розподіл навантаження на здорові репліки відбувається до фактичного OOM, що підвищує стійкість сервісу.

На 4-й годині моделюється відмова одного вузла. У централізованій схемі кількість активних pod'ів різко падає (наприклад, з 6 до 4), а потім повільно відновлюється до попереднього рівня, що відображає сумарну затримку перепланування pod'ів і додавання нового вузла Cluster Autoscaler'ом. У P2P-системі спад менш глибокий, а відновлення до попереднього рівня масштабу відбувається значно швидше, тобто час відновлення істотно менший.

На рис. 4 зображено зміну максимального завантаження CPU по вузлах у часі для обох підходів. На осі ординат – максимальний відсоток завантаження серед усіх вузлів у кожний момент часу; пунктирною лінією показано рівень 100%, що відповідає повному насиченню CPU

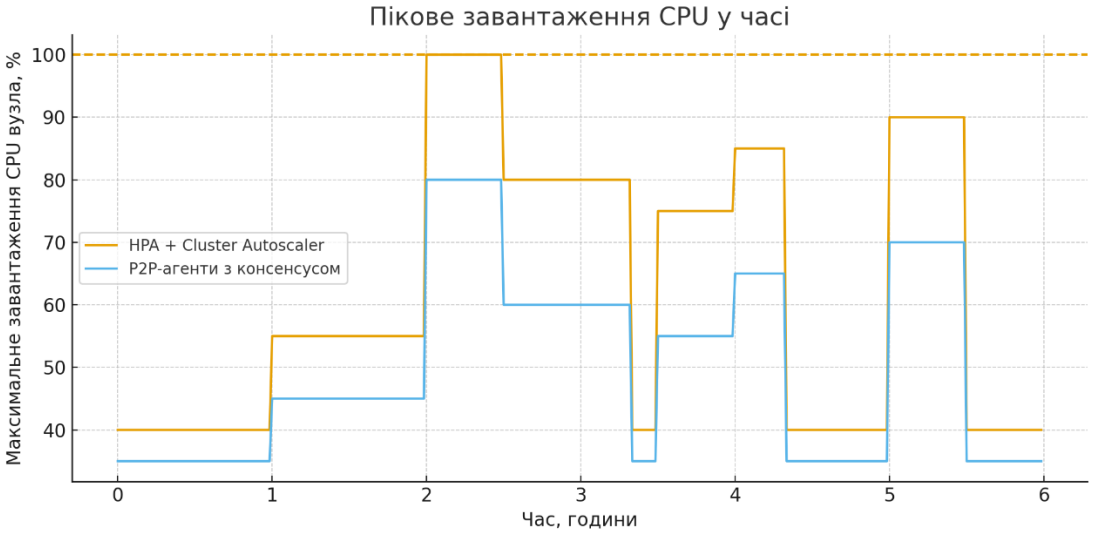

Рис. 4. Зміна максимального завантаження CPU

У фазі плавного навантаження (до 2-ї години) HPA+CA доводить пікове завантаження до приблизно 55%, тоді як P2P-система утримує його на рівні 45%. Це вже свідчить про більш рівномірний розподіл навантаження у P2P-випадку: агенти завчасно додають репліки і розкладають трафік, не доводячи вузли до граничних станів. Однак найбільш виразні відмінності виникають під час сплеску навантаження.

Під час пікового стрибка HPA миттєво виходить на 100% завантаження окремих вузлів, що означає повне насичення обчислювальних ресурсів, зростання затримок та підвищений ризик каскадної деградації при будь-яких додаткових коливаннях навантаження. Натомість P2P-система обмежує пік на рівні 80% завдяки швидшому масштабуванню та оперативному перерозподілу трафіку, не доводячи вузли до критичного режиму. Це важлива практична перевага: мінімізація пікових перевантажень досягається за рахунок алгоритмічної ефективності, а не додаткових резервів апаратних ресурсів.

Після проходження піку централізований автоскейлер ще тривалий час утримує підвищене завантаження (80-85%), тоді як P2P швидше повертається до стабільної робочої зони (60-65%). Подібна картина спостерігається й при відмові вузла: у HPA залишкові вузли тимчасово перевантажуються до 85-90%, тоді як у P2P завдяки наявному резерву та колективній реакції агентів завантаження обмежується 65-70%.

Важливою характеристикою децентралізованої системи є латентність консенсусу – час між ініціацією керувальної пропозиції (запису до журналу) та її підтвердженням більшістю агентів (commit). На рис. 5 наведено часовий ряд латентності консенсусу P2P-системи протягом експерименту.

Рис. 5. Графік латентності консенсусу P2P-системи

У стаціонарних умовах (до 4-ї години) латентність тримається на рівні 0.2 с. Це означає, що агентська мережа здатна узгоджувати рішення про масштабування в суб-секундних масштабах, що набагато швидше, ніж типовий цикл HPA, прив'язаний до періодичних інтервалів збору метрик. Такий рівень швидкодії робить можливим квазі-онлайн керування, коли рішення приймаються практично в режимі «реального часу» з погляду хмарних додатків.

При моделюванні відмови вузла (4-та година) спостерігається короткочасний сплеск латентності до 1 с. Це відповідає фазі перевиборів лідера в протоколі Raft: агенти, що втратили контакт із попереднім лідером, запускають процедуру виборів, збирають голоси, обирають нового лідера й відновлюють нормальну реплікацію журналу. Після завершення цієї фази латентність падає назад до базового рівня 0.2-0.3 с, що свідчить про швидку конвергенцію після локального збою.

Найскладніший сценарій – мережеве розділення (5-та година). На цьому інтервалі латентність стрибкоподібно зростає до 5 с. Такий ефект є очікуваним: частина агентів більше не може досягнути кворуму у звичному складі, повторні спроби узгодження зустрічають тайм-аути, і в алгоритмі активізуються механізми обмеження частоти та локальної деградації. Однак ключовим є те, що навіть при такому зростанні затримки система не зупиняється – консенсус продовжує працювати у доступних підмножинах, а логіка агентів забезпечує безпечний перехід секцій кластера у автономний режим. Після відновлення зв'язку латентність стрімко повертається до базових 0.2-0.3 с., що демонструє відсутність довготривалої деградації алгоритму.

На рис. 6 наведено порівняння службового мережевого трафіку, що генерується механізмами автоскейлінгу протягом шестигодинного експерименту для обох підходів. У випадку централізованої схеми HPA+CA трафік складається переважно зі збору метрик та періодичних запитів контролерів автоскейлінгу [11]. У P2P-системі до цього додається інтенсивний обмін повідомленнями між агентами (heartbeat, пропозиції та підтвердження консенсусу, обмін прогнозами тощо).

Рис. 6. Порівняння службового мережевого трафіку

Згідно з оцінками, централізований підхід генерує приблизно 5 МБ службового трафіку за 6 годин, тоді як P2P – близько 20 МБ. Тобто децентралізована архітектура збільшує накладний трафік приблизно у 4 рази. Це логічний наслідок переходу від «один контролер – багато вузлів» до «багато агентів – багато агентів». Утім, в абсолютних величинах 20 МБ за 6 годин є незначною величиною для сучасних датацентрів та кластерних мереж, що не створює помітного навантаження на мережеву інфраструктуру й не конкурує з корисним трафіком додатків. З технічної точки зору це означає, що мережева ціна децентралізації відносно невелика, а виграш у стійкості, швидкодії та якості керування ресурсами, переважує цей overhead у більшості сценаріїв. За потреби P2P-протокол може бути додатково оптимізований: наприклад, зменшення частоти heartbeat [12] у спокійному режимі, агрегація повідомлень, адаптивний gossip тощо.

Висновки

У ході дослідження були послідовно розв'язані всі задачі, сформульовані на етапі постановки проблеми. Було повністю реалізовано та експериментально підтверджено, що децентралізована самоадаптивна архітектура керування ресурсами з P2P-взаємодією та консенсусом забезпечує вищу стійкість, швидкість реакції та ефективність розподілу навантаження у хмарних обчислювальних системах порівняно з класичними централізованими механізмами масштабування Kubernetes (HPA та Cluster Autoscaler).

У роботі вперше запропоновано формальну модель самоадаптивного агентного керування ресурсами для cloud-native середовища, у якій кожен вузол кластера отримує власний автономний локальний контролер зі внутрішньою структурою типу MAPE та механізмами прогнозування, P2P-синхронізації та консенсусного прийняття рішень. Створено математичну та алгоритмічну основу для локальної та глобальної координації рішень щодо масштабування сервісів і перерозподілу навантаження в умовах часткових відмов, мережевих розділень та неповноти телеметрії.

Детально спроєктовано технічну архітектуру системи: описано структуру вузлового агента, механізм обміну P2P-повідомленнями, локальні алгоритми прогнозування, процедури формування пропозицій щодо масштабування, реалізацію консенсусу, а також механізми стабілізації та запобігання небажаним коливанням масштабу. Окремо побудовано внутрішню модель відмов і мережевих сценаріїв, що дозволило адекватно відтворити поведінку реальних кластерів.

Отримані результати демонструють суттєві переваги децентралізованої архітектури. Середній час реакції на раптові піки навантаження у P2P-системі скоротився в 3-5 разів порівняно з HPA, а пікове завантаження CPU зменшилося з 100% до 80%, що дозволяє уникати фаз повного насичення та деградації сервісу. Стійкість до відмов вузлів виявилася істотно вищою: P2P-система відновлювала цільовий масштаб у 2.5 рази швидше, а суттєве перевантаження CPU після відмов було обмежено. У разі «network partition»

централізований автоскейлер втрачає здатність коректно реагувати, у той час як P2P-агенти продовжують масштабування всередині зв'язних компонентів графа та утримують паузу в діапазоні оптимальних робочих ресурсів. Площа під кривою перевантаження CPU (інтегральний час роботи вище порогового рівня 80%) для HPA виявилася більшою у 2.7 раза, що прямо відображає вищу операційну надійність P2P-системи.

Таким чином, у дослідженні показано, що запропонована децентралізована модель керування ресурсами забезпечує вищу адаптивність, відмовостійкість та ефективність у порівнянні з класичними централізованими механізмами Kubernetes, особливо в умовах непередбачуваних піків навантаження, часткових відмов та неповної телеметрії. Це відкриває перспективу практичного застосування таких систем у промислових «cloud-native» платформах, а також можливість подальшого розширення моделі до гібридних варіантів із ієрархічним керуванням, інтеграцією підкріплювального навчання та динамічною реконфігурацією P2P-топології.

Література

1. Mykhailichenko, I., Liashenko, O. Informer: Resource usage forecasting model in cloud computing using Informer architecture.. Management Information System and Devises, 1(186), 17-28, 2025. DOI: 10.30837/0135-1710.2025.186.017.

2. Song B., Guo B., Hu W., Zhang Z., Zhang N., Bao J., Wang J., Xin J. Transformer-Based Time-Series Forecasting for Telemetry Data in an Environmental Control and Life Support System of Spacecraft. Electronics, 14 (3): 459, 2025. DOI: 10.3390/electronics14030459.

3. Smendowski M., et al. Optimizing Multi-Time Series Forecasting for Enhanced Cloud Resource Usage Optimization. Journal of Cloud Computing, 2024. DOI: 10.1016/j.cloud.2024.

4. Liu D., et al. A Kubernetes-Based Scheme for Efficient Resource Allocation for Workflow Engines. Computers & Industrial Engineering, 2025. DOI: 10.1016/j.cie.2025.

5. Taheri J., et al. Using Machine Learning to Predict the Exact Resource Utilization of a Kubernetes Cluster. Preprint, 2023. URL: https://www.diva-portal.org/smash/get/diva2%3A1869851/FULLTEXT01.pdf.

6. Lian L., Li Y., Han S., Meng R., Wang S., Ming W. Artificial Intelligence-Based Multiscale Temporal Modeling for Anomaly Detection in Cloud Services. Preprint, 2025. DOI: 10.48550/arXiv.2508.14503.

7. Lingrui Yu. DTAAD: Dual TCN-Attention Networks for Anomaly Detection in Multivariate Time Series Data. Preprint, 2023. DOI: 10.48550/arXiv.2302.10753.

8. Chakraborty S., Heintz F. Enhancing Time Series Forecasting with Fuzzy Attention-Integrated Transformers. Preprint, 2025. DOI: 10.48550/arXiv.2504.00070.

9. Tuli S., Casale G., Jennings N. TranAD: Deep Transformer Networks for Anomaly Detection in Multivariate Time Series Data. Preprint, 2022. DOI: 10.48550/arXiv.2201.07284.

10. Qin Y., Zeng P., Yan R., Zhang Y., Li B., Ding J. PatchTST: "A Time Series is Worth 64 Words" – Long-term Forecasting with Transformers. ICLR Workshop, 2023. DOI: 10.48550/arXiv.2211.14730.

11. Wen Q., et al. Transformers in Time Series: A Survey. IJCAI, 2022. DOI: 10.24963/ijcai.2023/759.

12. Liu Y., Wu H., Wang J., Long M. Non-stationary Transformers: Exploring the Stationarity in Time Series Forecasting. Proc. of AAAI, 2022. DOI: 10.48550/arXiv.2205.14415.

References

1. Mykhailichenko, I., Liashenko, O. Informer: Resource usage forecasting model in cloud computing using Informer architecture.. Management Information System and Devises, 1(186), 17-28, 2025. DOI: 10.30837/0135-1710.2025.186.017.

2. Song B., Guo B., Hu W., Zhang Z., Zhang N., Bao J., Wang J., Xin J. Transformer-Based Time-Series Forecasting for Telemetry Data in an Environmental Control and Life Support System of Spacecraft. Electronics, 14 (3): 459, 2025. DOI: 10.3390/electronics14030459.

3. Smendowski M., et al. Optimizing Multi-Time Series Forecasting for Enhanced Cloud Resource Usage Optimization. Journal of Cloud Computing, 2024. DOI: 10.1016/j.cloud.2024.

4. Liu D., et al. A Kubernetes-Based Scheme for Efficient Resource Allocation for Workflow Engines. Computers & Industrial Engineering, 2025. DOI: 10.1016/j.cie.2025s.

5. Taheri J., et al. Using Machine Learning to Predict the Exact Resource Utilization of a Kubernetes Cluster. Preprint, 2023. URL: https://www.diva-portal.org/smash/get/diva2%3A1869851/FULLTEXT01.pdf.

6. Lian L., Li Y., Han S., Meng R., Wang S., Ming W. Artificial Intelligence-Based Multiscale Temporal Modeling for Anomaly Detection in Cloud Services. Preprint, 2025. DOI: 10.48550/arXiv.2508.14503.

7. Lingrui Yu. DTAAD: Dual TCN-Attention Networks for Anomaly Detection in Multivariate Time Series Data. Preprint, 2023. DOI: 10.48550/arXiv.2302.10753.

8. Chakraborty S., Heintz F. Enhancing Time Series Forecasting with Fuzzy Attention-Integrated Transformers. Preprint, 2025. DOI: 10.48550/arXiv.2504.00070.

9. Tuli S., Casale G., Jennings N. TranAD: Deep Transformer Networks for Anomaly Detection in Multivariate Time Series Data. Preprint, 2022. DOI: 10.48550/arXiv.2201.07284.

10. Qin Y., Zeng P., Yan R., Zhang Y., Li B., Ding J. PatchTST: "A Time Series is Worth 64 Words" – Long-term Forecasting with Transformers. ICLR Workshop, 2023. DOI: 10.48550/arXiv.2211.14730.

11. Wen Q., et al. Transformers in Time Series: A Survey. IJCAI, 2022. DOI: 10.24963/ijcai.2023/759.

12. Liu Y., Wu H., Wang J., Long M. Non-stationary Transformers: Exploring the Stationarity in Time Series Forecasting. Proc. of AAAI, 2022. DOI: 10.48550/arXiv.2205.14415.