УДК 004.75:004.42 DOI https://doi.org/10.35546/kntu2078-4481.2025.4.3.11

Г. А. ЗАВГОРОДНЯ
кандидат технічних наук, доцент,
доцент кафедри обчислювальної техніки
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
ORCID: 0000-0001-8523-1761

В. В. ЗАВГОРОДНІЙ
доктор технічних наук, професор,
професор кафедри обчислювальної техніки
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
ORCID: 0000-0002-8347-7183

РОЗРОБКА МАСШТАБОВАНОЇ РОЗПОДІЛЕНОЇ АРХІТЕКТУРИ ДЛЯ МАСОВИХ БАГАТОКОРИСТУВАЦЬКИХ ОНЛАЙН-СИСТЕМ

У сучасній ігровій індустрії та суміжних сферах цифрових технологій дедалі зростає потреба у створенні систем, здатних забезпечувати стабільне функціонування в умовах надзвичайно великої кількості одночасних користувачів. Масові багатокористувацькі онлайн-системи (ММО) – це складні програмно-апаратні комплекси, у яких необхідно гарантувати не лише швидкий обмін даними між клієнтами, а й узгодженість станів, масштабованість, відмовостійкість і мінімальну затримку реакції сервера. Традиційні централізовані архітектури не можуть задовольнити ці вимоги, оскільки їх продуктивність обмежується апаратними ресурсами одного вузла, що призводить до зниження якості обслуговування користувачів при пікових навантаженнях.

Запропоновано алгоритм динамічного балансування, який використовує комбінацію локальних метрик завантаження та глобального аналізу стану системи. На основі модулю машинного навчання (MLLP) реалізовано прогнозування майбутніх пікових навантажень, що дозволяє попередньо перерозподіляти ресурси між серверами. Програмну реалізацію алгоритму виконано на мові Python із використанням технологій багатопроцесорної обробки та асинхронних черг повідомлень.

Порівняльний аналіз результатів показав, що розроблена архітектура забезпечує зниження середньої латентності на 57% у порівнянні з класичними централізованими ММО-моделями та підвищення ефективності використання обчислювальних ресурсів на понад 230%. Отримані результати підтверджують доцільність застосування розподілених архітектур у сучасних ігрових середовищах і платформах реального часу.

Практичне значення дослідження полягає у можливості використання запропонованої архітектури для побудови високонавантажених серверів ММО-ігор, симуляторів та інтерактивних освітніх або віртуальних середовищ. Подальші дослідження планується спрямувати на вдосконалення моделей синхронізації станів, розроблення гібридних підходів до балансування та інтеграцію систем самонавчання для автономного керування ресурсами.

Ключові слова: *ММО, розподілені системи, масштабованість, балансування навантаження, архітектура серверів, машинне навчання.*

G. A. ZAVHORODNIA
Ph.D., Associate Professor,
Associate Professor at the Department of Computer Engineering
National Technical University of Ukraine
“Igor Sikorsky Kyiv Polytechnic Institute”
ORCID: 0000-0001-8523-1761

V. V. ZAVHORODNII
Doctor of Sciences, Professor,
Professor at the Department of Computer Engineering
National Technical University of Ukraine
“Igor Sikorsky Kyiv Polytechnic Institute”
ORCID: 0000-0002-8347-7183

DEVELOPMENT OF A SCALABLE DISTRIBUTED ARCHITECTURE FOR MASSIVELY MULTIPLAYER ONLINE SYSTEMS

In modern gaming and digital technology industries, there is an increasing demand for systems capable of maintaining stable performance under extremely high user concurrency. Massively multiplayer online (MMO) systems are complex software and hardware infrastructures that must guarantee not only rapid data exchange between clients but also consistency of shared states, scalability, fault tolerance, and minimal response latency. Traditional centralized architectures fail to meet these requirements because their performance is limited by the hardware resources of a single node, resulting in service degradation under peak loads.

A dynamic balancing algorithm is proposed that combines local load metrics with global system state analysis. Using the Machine Learning Load Predictor (MLLP) module, the algorithm predicts future peak loads, allowing preemptive redistribution of computational resources between servers. The software implementation of the algorithm was performed in Python using multiprocessing and asynchronous message queue technologies.

A comparative performance analysis showed that the developed architecture reduces average latency by 57% compared to conventional centralized MMO architectures and increases computational resource utilization efficiency by more than 230%. These results confirm the feasibility of applying distributed architectures in modern real-time gaming and simulation platforms.

The practical significance of the research lies in the potential use of the proposed architecture for building high-load MMO servers, distributed simulations, and interactive educational or virtual environments. Future research will focus on improving state synchronization models, developing hybrid balancing strategies, and integrating self-learning systems for autonomous resource management.

Key words: *MMO, distributed systems, scalability, load balancing, server architecture, machine learning.*

Постановка проблеми

У сучасній ігровій індустрії масові багатокористувацькі онлайн-системи (*MMO*) стають одними з найбільш ресурсомістких об'єктів програмної інженерії. Їхня архітектура вимагає одночасної обробки великої кількості взаємодій між користувачами в режимі реального часу, що створює критичні вимоги до масштабованості, продуктивності та стійкості системи [1; 2; 3]. Традиційні централізовані підходи, коли основні обчислення виконуються на одному або обмеженій кількості серверів, вичерпали свій потенціал через затримки, перевантаження мережевих каналів і складність балансування навантаження [4].

Розподілені архітектури *MMO* дають змогу динамічно масштабувати обчислювальні ресурси, розподіляти навантаження між географічно віддаленими вузлами та забезпечувати відмовостійкість під час пікових навантажень. Проте їх ефективна реалізація пов'язана з низкою проблем – узгодженням станів між серверами, мінімізацією затримок синхронізації, забезпеченням цілісності даних і підтриманням стабільної швидкодії при різкому зростанні кількості гравців [5; 6].

Актуальність проблеми зумовлена необхідністю розробки архітектурних моделей, здатних адаптивно перерозподіляти обчислювальні ресурси відповідно до реального навантаження. Для цього доцільно використовувати сучасні підходи математичного моделювання, машинного навчання та процедурної генерації даних, які вже довели свою ефективність у суміжних завданнях [7; 8].

Таким чином, постає науково-прикладне завдання розроблення масштабованої розподіленої архітектури для *MMO*-систем реального часу, яка забезпечує стабільну роботу при високих навантаженнях, автоматичне балансування та адаптивне розподілення обчислювальних процесів.

Аналіз останніх досліджень і публікацій

Розвиток масових багатокористувацьких онлайн-систем супроводжується активними дослідженнями у сфері побудови масштабованих архітектур, здатних підтримувати динамічне балансування навантаження між великою кількістю клієнтських підключень. У роботах останніх років спостерігається тенденція переходу від централізованих архітектур до гібридних моделей із використанням мікросервісів, контейнеризації та серверлес-підходів [2; 7]. Це дає змогу ефективніше розподіляти обчислювальні ресурси та знижувати затримки взаємодії між вузлами системи.

Зокрема, дослідження [1] пропонує підхід до розподілу обчислювального навантаження у реальному часі на основі глибокого навчання з підкріпленням, що дозволяє прогнозувати поведінку гравців та адаптивно балансувати ресурси серверів. У [4] розроблено алгоритм латентно-орієнтованого балансування для *MMO*, який мінімізує мережеві затримки та забезпечує стабільну частоту оновлення кадрів. Дослідження [6] описує архітектуру на базі периферійних (*edge*) обчислень, що забезпечує зменшення часу реакції системи за рахунок наближення обчислень до користувачів.

Окрему увагу науковці приділяють питанням математичного моделювання та формального аналізу процесів у розподілених архітектурах. У роботі [10] наведено приклади формального моделювання взаємодій між вузлами в розподілених середовищах, що дозволяє оцінювати стабільність і масштабованість систем ще на етапі

проектування. Такі підходи є важливою теоретичною основою для побудови архітектур MMO, які мають адаптивно реагувати на зміну ігрового навантаження.

Важливий внесок у дослідження суміжних технологій зроблено українськими авторами. Зокрема, у роботі [9] запропоновано метод автоматичної генерації контенту на основі процедурних алгоритмів, що може бути застосований для динамічного розширення *MMO*-середовищ без ручного втручання розробника. У [10] представлено методи математичного моделювання для формального дослідження розподілених процесів, що є базою для побудови аналітичних моделей масштабування. Робота [11] присвячена методам створення штучних текстур із заданими параметрами, що має потенціал для процедурного формування візуальних об'єктів у розподіленому середовищі. У [12] розглянуто підхід до виявлення аномалій за допомогою машинного навчання, який може бути інтегрований у систему моніторингу навантаження для динамічного балансування серверних ресурсів.

Аналіз сучасних публікацій свідчить, що існують окремі рішення для часткових аспектів проблеми – генерації контенту, балансування навантаження чи математичного моделювання. Водночас відсутня інтегрована модель, яка б поєднувала ці підходи в єдину розподілену архітектуру *MMO*-системи з адаптивним масштабуванням у режимі реального часу. Розроблення такої архітектури становить актуальну наукову задачу даної роботи.

Формулювання мети дослідження

Метою даного дослідження є розроблення масштабованої розподіленої архітектури для масових багатокористувацьких онлайн-систем, здатної забезпечувати стабільну роботу в умовах нерівномірного навантаження та великої кількості одночасних користувачів. Для досягнення цієї мети передбачено побудову математичної моделі балансування ресурсів, розроблення алгоритму динамічного розподілу обчислень між вузлами системи та експериментальну перевірку ефективності запропонованого підходу порівняно з відомими базовими архітектурами.

Викладення основного матеріалу дослідження

Сучасна масова багатокористувацька онлайн-система являє собою складну динамічну екосистему, у якій одночасно взаємодіють тисячі клієнтів, підключених до серверного кластера. Основними вимогами до архітектури такої системи є низька латентність, стійкість до пікових навантажень, балансування ресурсів і відмовостійкість. Традиційна серверно-клієнтська модель при цьому має суттєві обмеження – один центральний сервер стає «вузьким місцем» під час масових підключень.

Запропонована архітектура ґрунтується на розподіленому обчислювальному ядрі (*Distributed Core*), яке складається з незалежних вузлів (*Node_i*), пов'язаних через брокер повідомлень (*gRPC, RabbitMQ*) і системи спільного стану (*State Sync Service*). Кожен вузол відповідає за певну географічну або логічну область віртуального світу (*Zone Sharding*). Вузли взаємодіють через подієву шину (*Event Bus*), що забезпечує узгодженість станів між сегментами світу.

Схематично модель можна подати як множину вузлів:

$$N = \{n_1, n_2, \ldots, n_k\},$$

де n_i – i -тий вузол (сервер або процес), який бере участь у розподіленій архітектурі; k – загальна кількість вузлів у системі, $k \in \mathbb{N}$.

Кожен вузол n_i виконує функції обробки подій (*game events, network events* тощо), синхронізації станів із сусідніми вузлами та балансування навантаження між локальними процесами або кластерами.

Взаємодія між вузлами описується квадратною матрицею розмірністю $k \times k$, яка визначає наявність зв'язків між вузлами:

$$C = \left[c_{ij} \right], \qquad i, j = 1, 2, \ldots, k,$$

де $c_{ij} = 1$, якщо вузол n_i безпосередньо обмінюється даними з вузлом n_j; $c_{ij} = 0$, якщо прямого обміну даними між вузлами n_i та n_j немає; для всіх $i : c_{ii} = 0$, оскільки вузол не має зв'язку сам із собою.

Таким чином, матриця C є матрицею суміжності, яка визначає топологію зв'язків у розподіленій архітектурі. На її основі можна обчислювати ступінь зв'язності кожного вузла:

$$d_i = \sum_{j=1}^{k} c_{ij},$$

де d_i – ступінь зв'язності i -го вузла, що показує кількість сусідів, із якими він взаємодіє.

Математична модель балансування навантаження. Для формалізації процесу розподілу обчислювальних ресурсів у запропонованій архітектурі побудуємо математичну модель балансування навантаження, що описує взаємодію між вузлами системи та розподіл вхідного потоку запитів. Нехай:

$\lambda_i(t)$ – інтенсивність вхідних подій для вузла i в момент часу t; μ_i – пропускна здатність вузла i; $\rho_i(t) = \dfrac{\lambda_i(t)}{\mu_i}$ – коефіцієнт завантаження вузла.

Система вважається стабільною, якщо:

$$\rho_i(t) < 1, \quad \forall_i \in N .$$

Для досягнення стабільності застосовується динамічне перенаправлення навантаження між вузлами за допомогою функції перерозподілу:

$$\Delta\lambda_i(t) = \alpha \cdot \left(\rho_i(t) - \rho_j(t)\right),$$

де α – коефіцієнт чутливості балансування $(0 < \alpha \le 1)$.

Сумарне навантаження системи:

$$\Lambda(t) = \sum_{i=1}^{k} \lambda_i(t),$$

а середній рівень використання ресурсів:

$$R(t) = \frac{1}{k} \sum_{i=1}^{k} \rho_i(t) .$$

Завдання оптимального балансування формулюється як мінімізація дисперсії завантаження вузлів:

$$min\sigma_\rho^2 = \frac{1}{k} \sum_{i=1}^{k} (\rho_i - R)^2 .$$

Таким чином, алгоритм має прагнути до рівномірного завантаження серверів при збереженні мінімальної латентності обробки.

Алгоритм масштабування та міграції сесій. У межах запропонованої архітектури реалізовано механізм динамічного масштабування, який автоматично активує або деактивує серверні вузли відповідно до рівня поточного та прогнозованого навантаження. Для цього використовується предиктор, який оцінює майбутню інтенсивність запитів користувачів на основі короткострокової моделі прогнозування. Такий підхід дозволяє забезпечити адаптивну реакцію системи на пікові навантаження та мінімізувати затрати ресурсів у періоди низької активності. Прогнозне значення навантаження на момент часу $t + \Delta t$ визначається рівнянням:

$$\hat{\lambda}(t + \Delta t) = \beta_0 + \beta_1 \lambda(t) + \beta_2 \frac{d\lambda}{dt},$$

де $\hat{\lambda}(t + \Delta t)$ – прогнозована інтенсивність вхідних запитів в момент часу $t + \Delta t$; $\lambda(t)$ – поточна інтенсивність запитів в момент часу t; $\frac{d\lambda}{dt}$ – похідна інтенсивності запитів за часом, що характеризує швидкість зміни навантаження; $\beta_0, \beta_1, \beta_2$ – коефіцієнти моделі, які визначаються шляхом регресійного навчання на основі історичних даних про трафік; Δt – часовий інтервал прогнозування.

Після обчислення прогнозу система порівнює його із граничним порогом навантаження $\mu_{threshold}$, який визначає максимальну допустиму інтенсивність запитів для одного вузла:

$$\hat{\lambda}(t + \Delta t) > \mu_{threshold} \Rightarrow \textit{ініціалізація нового вузла} .$$

У випадку, якщо прогнозоване навантаження перевищує поріг $\mu_{threshold}$, система ініціює створення нового серверного вузла через механізм контейнеризації (*Docker* або *Kubernetes*). Новий вузол реєструється у балансувальнику навантаження та отримує частину активних сесій користувачів. Якщо ж навантаження знижується, і виконується умова

$$\hat{\lambda}(t + \Delta t) > \mu_{threshold} - \varepsilon ,$$

де ε – гістерезис для уникнення частого перемикання станів, то надлишкові вузли вимикаються за допомогою алгоритму *graceful shutdown*, який передбачає поступову міграцію активних сесій до інших вузлів без розриву з'єднань.

Процес міграції сесій відбувається за схемою:

1. Оповіщення балансувальника про майбутнє відключення вузла.
2. Перенаправлення нових з'єднань до інших вузлів.
3. Передача активних сесій через протокол обміну станами.
4. Вимкнення вузла після завершення всіх поточних транзакцій.

Таким чином, алгоритм масштабування та міграції сесій забезпечує еластичність системи, мінімізуючи час відгуку та підтримуючи стабільну якість обслуговування користувачів навіть при різких змінах навантаження.

З метою експериментальної перевірки ефективності математичної моделі та алгоритму масштабування розроблено їх прототипну реалізацію засобами *Python*. Наведений нижче код відображає основні логічні компоненти

алгоритму, включаючи оцінювання навантаження, прогнозування та прийняття рішень щодо активації або деактивації вузлів.

```
import numpy as np
class LoadBalancer:
    def __init__(self, nodes, alpha=0.3):
        self.nodes = np.array(nodes, dtype=float)
        self.alpha = alpha
    def balance(self):
        mean_load = np.mean(self.nodes)
        delta = self.alpha * (self.nodes - mean_load)
        self.nodes -= delta  # зменшуємо перевантаження
        return self.nodes
# Приклад використання
loads = [0.8, 0.4, 0.6, 0.9, 0.3]  # поточні коефіцієнти завантаження вузлів
balancer = LoadBalancer(loads)
new_loads = balancer.balance()
print("Скориговане навантаження:", new_loads)
```

Наведений фрагмент реалізації ілюструє базову концепцію децентралізованого балансування навантаження, що забезпечує мінімізацію відхилень між рівнями завантаження серверних вузлів. Запропонований підхід може бути розширено для інтеграції з *API* систем оркестрації контейнерів, де параметри α та граничні умови масштабування визначаються динамічно на основі прогнозної моделі навантаження.

Модель синхронізації станів. Стан ігрового світу в розподіленій *MMO*-системі повинен залишатися цілісним і узгодженим незалежно від кількості активних обчислювальних вузлів. Оскільки синхронна передача даних між усіма вузлами створює надмірне мережеве навантаження, у даній роботі використано модель часткової узгодженості (*eventual consistency*), за якої оновлення стану поширюються з часовою затримкою σ, але при цьому зберігається каузальна послідовність подій, тобто кожна дія виконується у правильному логічному порядку відносно попередніх. Стан кожного ігрового об'єкта i в момент часу t описується вектором стану:

$$S_i(t) = \left(p_i(t), v_i(t), h_i(t)\right),$$

де $p_i(t)$ –позиція об'єкта у просторі гри (координати (x, y, z)); $v_i(t)$ – вектор швидкості, який визначає напрям і величину руху; $h_i(t)$ – рівень «здоров'я» або інший параметр життєздатності об'єкта.

Зміна стану описується функцією оновлення:

$$S_i(t + \Delta t) = f\left(S_i(t), E_i(t)\right),$$

де $E_i(t)$ – множина подій, що впливають на об'єкт в момент часу t (колізії, атаки, взаємодії між гравцями, дії серверних скриптів); $f\left(S_i(t), E_i(t)\right)$ – детермінована або стохастична функція переходу стану, що визначає результат взаємодії.

Для узгодження станів між різними вузлами застосовується алгоритм типу *CRDT* (*Conflict-free Replicated Data Type*), який забезпечує ідемпотентність і комутативність операцій при асинхронному оновленні. Це означає, що порядок отримання повідомлень не впливає на кінцевий стан системи – усі вузли сходяться до одного узгодженого значення S_i^* після завершення обміну повідомленнями:

$$\lim_{t \to \infty} S_i^{(n)}(t) = S_i^*, \quad \forall n \in N,$$

де $S_i^{(n)}(t)$ – стан об'єкта i, відтворений на вузлі n, а N –множина всіх вузлів системи.

Таким чином, модель синхронізації на основі *CRDT* дозволяє уникнути блокувань і зберегти логічну цілісність ігрового середовища навіть при тимчасових мережевих розбіжностях або втраті пакетів. Це суттєво підвищує масштабованість і стійкість *MMO*-систем до асинхронності в обміні даними.

Механізм самонавчання для адаптації. Для забезпечення автономної адаптації системи до динамічних і непередбачуваних змін навантаження в архітектуру вводиться компонент *MLLP*. Основна мета цього модуля полягає у побудові адаптивної моделі, що в реальному часі навчається залежності між поточним станом кластеру та очікуваним рівнем навантаження. Такий підхід дозволяє зменшити кількість помилкових рішень при масштабуванні та забезпечує більш стабільну роботу всієї *MMO*-системи під час пікових навантажень. Модель *MLLP* реалізується у вигляді регресійної нейронної мережі, яка обчислює прогнозне значення навантаження за формулою:

$$y = f\left(W_x + b\right),$$

де $x \in \mathbb{R}^n$ – вектор ознак, що містить поточні характеристики системи (середнє завантаження процесора *CPU*, кількість активних з'єднань, середню мережеву затримку, використання пам'яті тощо); $W \in \mathbb{R}^{m \times n}$ – матриця вагових коефіцієнтів нейронної мережі; $b \in \mathbb{R}^m$ – вектор зсуву, який зміщує результати активації; $f\left(W_x + b\right)$ – активаційна функція (*ReLU*, *tanh*), що забезпечує нелінійність моделі; $y \in \mathbb{R}$ – прогнозоване значення навантаження системи у наступному часовому інтервалі.

На основі прогнозу y модуль прийняття рішень визначає, чи необхідно активувати нові серверні вузли, або, навпаки, вимкнути надлишкові ресурси. Цей процес тісно інтегрований із блоком балансування навантаження, описаним у попередніх підрозділах. Навчання моделі *MLLP* відбувається в онлайновому режимі, тобто ваги оновлюються безпосередньо після кожного циклу вимірювання навантаження. Процес оновлення параметрів здійснюється за допомогою методу стохастичного градієнтного спуску (*SGD*):

$$W_{t+1} = W_t - \eta \nabla_W L\left(y, \hat{y}\right),$$

де W_t – матриця ваг на ітерації t; η – швидкість навчання, що визначає величину кроку оновлення; $L\left(y, \hat{y}\right)$ – функція втрат, яка вимірює відхилення між реальним навантаженням y та прогнозом $\hat{y}$.; $\nabla_W L\left(y, \hat{y}\right)$ – градієнт функції втрат за параметрами W.

Для стабільності навчання параметри η та розмір вибірки для оновлення ваг можуть динамічно змінюватися залежно від рівня флуктуацій у вхідних даних. Це дозволяє *MLLP*-модулю не лише реагувати на поточний стан системи, але й поступово вдосконалювати точність прогнозування під час роботи кластера.

Таким чином, механізм самонавчання забезпечує адаптивне масштабування *MMO*-архітектури, де процеси балансування, прогнозування та керування ресурсами утворюють єдиний інтелектуальний контур зворотного зв'язку.

Експериментальні результати. Для оцінки ефективності запропонованого підходу проведено експериментальне моделювання системи з 1000 активних користувачів, підключених до 10 серверних вузлів. Було порівняно три архітектури: централізовану (монолітну), розподілену без прогнозування та запропоновану архітектуру з використанням *MLLP* та динамічного балансування (табл. 1).

Таблиця 1

Порівняльні експериментальні результати різних архітектур системи для 1000 користувачів

№	Архітектура	Середня латентність, мс	Максимальне навантаження (користувачів / сервер)	Середнє *CPU*-завантаження, %	Відхилення навантаження (σ_ρ)	Ефективність, %
1	Централізована (монолітна)	220	250	95	0.31	100
2	Розподілена без прогнозу-вання	140	450	82	0.19	157
3	Запропонована архітектура (з *MLLP* і дина-мічним балан-суванням)	95	700	74	0.08	232

Результати експерименту показують, що застосування адаптивного балансування на основі прогнозної моделі дозволяє зменшити середню латентність на 57% та підвищити продуктивність системи більш ніж удвічі порівняно з класичними архітектурами. Крім того, запропонована система демонструє значно менше відхилення навантаження між вузлами $\sigma_\rho = 0.08$, що свідчить про ективне горизонтальне масштабування та інтелектуальне балансування ресурсів.

Висновки

У результаті проведеного дослідження розроблено масштабовану розподілену архітектуру для масових багатокористувацьких онлайн-систем, що забезпечує підвищену стійкість до навантаження, зниження латентності та ефективне використання обчислювальних ресурсів.

Запропоновано математичну модель балансування навантаження між вузлами, яка мінімізує дисперсію завантаження та підтримує стабільність системи. Реалізовано алгоритм динамічного перерозподілу запитів із урахуванням прогнозованого стану системи, що дозволяє адаптувати архітектуру до змін трафіку в реальному часі.

Додаткове впровадження модуля машинного навчання дало змогу підвищити ефективність балансування на понад 230% у порівнянні з класичними централізованими рішеннями, а середню затримку відповіді зменшити майже вдвічі.

Практичне значення роботи полягає у можливості використання запропонованої архітектури для розроблення ігрових серверів нового покоління, розподілених симуляцій та інтерактивних онлайн-платформ із високою кількістю користувачів.

Перспективами подальших досліджень є розширення системи за рахунок гібридних моделей синхронізації станів, інтеграція навчання з підкріпленням для автономного керування ресурсами та оцінювання енергетичної ефективності масштабованих *MMO*-середовищ.

Список використаної літератури

1. Barros e Sá G.C., Madeira C.A.G. Deep reinforcement learning in real-time strategy games. *Expert Systems with Applications*. 2021. DOI: https://doi.org/10.1007/s10489-024-06220-4

2. Rahmani A.M., Liljeberg P., Preden J.-S., Jantsch A. *Fog Computing in the Internet of Things: Intelligence at the Edge*. *Springer International Publishing*, 2018. ISBN 978 3 319 57638 1. DOI: https://doi.org/10.1007/978-3-319-57639-8

3. Kasenides N., Paspallis N. Athlos: A Framework for Developing Scalable MMOG Backends on Commodity Clouds. *Software*. 2022. Vol. 1, No. 1, pp. 107-145. DOI: https://doi.org/10.3390/software1010006

4. García Fernández E.J., García Puche E.J., Jabba Molinares D. Dynamic Low-Latency Load Balancing Model to Improve Quality of Experience in a Hybrid Fog and Edge Architecture for Massively Multiplayer Online (MMO) Games. *Applied Sciences*. 2025. Vol. 15, No. 12, Article 6379. DOI: https://doi.org/10.3390/app15126379

5. Zhou Y., Li D., Gao F. Optimal Synchronization Control for Heterogeneous Multi-Agent Systems: Online Adaptive Learning Solutions. 2020. DOI: https://doi.org/10.48550/arXiv.2011.05663

6. Wang N., Varghese B., Matthaiou M., Nikolopoulos D.S. ENORM: A Framework For Edge Node Resource Management. 2017. DOI: https://doi.org/10.48550/arXiv.1709.04061

7. Donkervliet J., Ron J., Li J., Iancu T., Abad C.L., Iosup A. Servo: Increasing the Scalability of Modifiable Virtual Environments Using Serverless Computing. 2023. *arXiv*. DOI: https://doi.org/10.48550/arXiv.2305.00032

8. Moreno-Vozmediano R., Huedo E., Montero R.S., Llorente I.M. AI-Driven Resource Allocation and Auto-Scaling of VNFs in Edge-5G-IoT Ecosystems. *Electronics*. 2025. Vol. 14, No. 9, Article 1808. DOI: https://doi.org/10.3390/electronics14091808

9. Завгородній В.В., Завгородня Г.А., Валявська Н.О., Адаменко В.С., Дороговцев Є.В., Несмачний П.В. Метод автоматичної генерації контенту на основі процедурних алгоритмів. *Вчені записки ТНУ ім. В. І. Вернадського. Серія: Технічні науки*. 2022. Т. 33(72), № 1, С. 91–96. DOI: https://doi.org/10.32838/2663-5941/2022.1/15

10. Завгородній В.В., Завгородня Г.А., Дроботович К.Є., Тенігін О.В., Шматко М.М. Математичне моделювання у методах формального дослідження. *Вчені записки ТНУ ім. В. І. Вернадського. Серія: Технічні науки*. 2021. Т. 32(71), № 6, С. 75–79. DOI: https://doi.org/10.32838/2663-5941/2021.6/12

11. Завгородній В.В., Завгородня Г.А., Демченко І.В., Крамаренко К.С., Шевченко І.О., Юрченко А.В. Метод створення штучних текстур із заданими параметрами. *Вчені записки ТНУ ім. В. І. Вернадського. Серія: Технічні науки*. 2022. Т. 33(72), № 2, С. 86–90. DOI: https://doi.org/10.32838/2663-5941/2022.2/14

12. Завгородній В.В., Завгородня Г.А., Валявська Н.О., Герасименко О.О., Калюжний О.В., Степовий А.В. Пошук аномалій у даних за допомогою машинного навчання. *Вчені записки ТНУ ім. В. І. Вернадського. Серія: Технічні науки*. 2022. Т. 33(72), № 3, С. 39–43. DOI: https://doi.org/10.32838/2663-5941/2022.3/06

References

1. Barros e Sá, G. C., & Madeira, C. A. G. (2021). Deep reinforcement learning in real-time strategy games. *Expert Systems with Applications*. https://doi.org/10.1007/s10489-024-06220-4

2. Rahmani, A. M., Liljeberg, P., Preden, J.-S., & Jantsch, A. (2018). *Fog Computing in the Internet of Things: Intelligence at the Edge. Springer International Publishing*. https://doi.org/10.1007/978-3-319-57639-8

3. Kasenides, N., & Paspallis, N. (2022). Athlos: A Framework for Developing Scalable MMOG Backends on Commodity Clouds. *Software*, 1(1), 107-145. https://doi.org/10.3390/software1010006

4. García Fernández, E. J., García Puche, E. J., & Jabba Molinares, D. (2025). Dynamic Low-Latency Load Balancing Model to Improve Quality of Experience in a Hybrid Fog and Edge Architecture for Massively Multiplayer Online (MMO) Games. *Applied Sciences*, 15(12), Article 6379. https://doi.org/10.3390/app15126379

5. Zhou, Y., Li, D., & Gao, F. (2020). Optimal Synchronization Control for Heterogeneous Multi-Agent Systems: Online Adaptive Learning Solutions. https://doi.org/10.48550/arXiv.2011.05663

6. Wang, N., Varghese, B., Matthaiou, M., & Nikolopoulos, D. S. (2017). ENORM: A Framework For Edge Node Resource Management. https://doi.org/10.48550/arXiv.1709.04061

7. Donkervliet, J., Ron, J., Li, J., Iancu, T., Abad, C. L., & Iosup, A. (2023). Servo: Increasing the Scalability of Modifiable Virtual Environments Using Serverless Computing. *arXiv*. https://doi.org/10.48550/arXiv.2305.00032

8. Moreno-Vozmediano, R., Huedo, E., Montero, R. S., & Llorente, I. M. (2025). AI-Driven Resource Allocation and Auto-Scaling of VNFs in Edge-5G-IoT Ecosystems. *Electronics*, 14(9), 1808. https://doi.org/10.3390/electronics14091808

9. Zavhorodnii, V. V., Zavhorodnia, H. A., Valiavska, N. O., Adamenko, V. S., Dorohovtsev, E. V., & Nesmachnyi, P. V. (2022). Method of automatic content generation based on procedural algorithms. *Vcheni Zapysky TNU im. V. I. Vernadskoho. Series: Technical Sciences*, 33(72), 91–96. https://doi.org/10.32838/2663-5941/2022.1/15 [in Ukrainian]

10. Zavhorodnii, V. V., Zavhorodnia, H. A., Drobotovych, K. E., Tenihin, O. V., & Shmatko, M. M. (2021). Mathematical modeling in formal research methods. *Vcheni Zapysky TNU im. V. I. Vernadskoho. Series: Technical Sciences*, 32(71), 75–79. https://doi.org/10.32838/2663-5941/2021.6/12 [in Ukrainian]

11. Zavhorodnii, V. V., Zavhorodnia, H. A., Demchenko, I. V., Kramarenko, K. S., Shevchenko, I. O., & Yurchenko, A. V. (2022). Method of creating artificial textures with specified parameters. *Vcheni Zapysky TNU im. V. I. Vernadskoho. Series: Technical Sciences*, 33(72), 86–90. https://doi.org/10.32838/2663-594 [in Ukrainian]

12. Zavhorodnii, V. V., Zavhorodnia, H. A., Valiavska, N. O., Herasymenko, O. O., Kalyuzhnyi, O. V., & Stepovyi, A. V. (2022). Anomaly detection in data using machine learning. *Vcheni Zapysky TNU im. V. I. Vernadskoho. Series: Technical Sciences*, 33(72), 39–43. DOI: https://doi.org/10.32838/2663-5941/2022.3/06 [in Ukrainian]

Дата першого надходження рукопису до видання: 29.11.2025
Дата прийнятого до друку рукопису після рецензування: 12.12.2025
Дата публікації: 31.12.2025