

ДОСЛІДЖЕННЯ ЗАТРИМОК РОБОТИ СЕРВЕРА ТЕЛЕМЕТРІЇ ДЛЯ ОФЛАЙН RPG-ВІДЕОІГОР

¹Вінницький національний технічний університет

Телеметрія є невіддільним елементом підтримки сучасного програмного забезпечення, що полягає у збиранні діагностичних даних для оперативного усунення технічних дефектів. Класичним прикладом цієї технології є відправка звітів про критичні помилки. Зокрема, у разі аварійної зупинки додатка система автоматично передає причини збою розробникам дампу пам'яті для аналізу. У ситуаціях, коли програма функціонує без постійного доступу до мережі, аналітика акумулюється локально. Передача цих даних на сервер ініціюється автоматично одразу після відновлення інтернет-з'єднання або за настання певних системних тригерів. Видавці відеоігор, особливо офлайн RPG-ігор, активно використовують телеметрію для дослідження балансу складності, геймплею та розуміння переваг своєї аудиторії. Для цих цілей необхідним є серверний додаток, розгорнутий у хмарних датацентрах, який має структуру бази даних, характерну для RPG-ігор, та здатний забезпечити надходження даних у режимі реального часу з низькими затримками. У межах дослідження розгорнуто експериментальний стенд на базі п'яти віртуальних машин з ОС Ubuntu GNU/Linux з використанням гіпервізора VMWare ESXi. Проведено порівняльний аналіз п'яти архітектурних конфігурацій, що поєднують нативне розгортання та Docker-контейнеризацію для API-сервера і бази даних MySQL. Методологія базується на сценаріях навантажувального тестування, що імітує пікові залпові навантаження, характерні для ігрових клієнтів у разі відновлення мережевого з'єднання. Встановлено, що найефективнішою для ігрового сервера телеметрії є архітектура з використанням мережевого простору хосту, тоді як стандартні операції читання та авторизації успішно реалізуються в межах ізольованих контейнерних мереж. Розроблений метод аналізу дозволяє зробити вибір оптимального серверного середовища, мінімізуючи мережеві затримки та забезпечуючи масштабованість інфраструктури для підтримки великої кількості одночасних підключень.

Ключові слова: телеметрія, відеоігри, затримки, сервер, хмара, GNU/Linux, Docker.

Вступ

Сьогодні телеметрія є складовою функціонування багатьох локальних додатків. Розробники деколи збирають інформацію діагностування з додатку на боці споживача. Розробникам така інформація потрібна для швидшого виявлення недоліків додатку. Користувач безпосередньо стикається з відправкою даних телеметрії у вигляді Crash Reporting, коли додаток аварійно завершує роботу. У такому випадку надсилається дампу пам'яті процесу та супутні дані, з якими працював додаток [1]. Отримання подібних дампув дозволяє розробникам швидко знайти причину збоїв та усунути її. Проте накопичення та надсилання даних телеметрії може відбуватися не лише під час аварійного завершення роботи додатку. Додатки, які переважно працюють офлайн, можуть локально збирати аналітичну інформацію про свою роботу, надсилаючи її в той момент, коли з'єднання з Інтернетом відновиться або у разі настання певних визначених умов [2].

Зазвичай розробники додатків повідомляють користувачів про правила конфіденційності під час збору та обробки даних, що вимагається законодавством різних країн. На території Європейського Союзу (ЄС) діє регламент GDPR. Усі постачальники програмного забезпечення, продукція яких використовується у країнах ЄС, зобов'язані його дотримуватись [3]. Компромісним рішенням у межах регламенту є принцип opt-out, коли користувач має право відмовитися від збирання і надсилання даних телеметрії. Іншим важливим правом, яке у межах GDPR має особа, є можливість вимагати знищення всіх своїх даних за вимогою. Тому під час проектування інструменти телеметрії мають передбачати дотримання всіх прав, гарантованих регламентом GDPR [3]. З часом збір даних



телеметрії втратив як мету лише аналіз причин критичних збоїв, розробники додатків стали використовувати ці аналітичні дані для оптимізації продуктивності роботи власних додатків, аналізу використання їх функціоналу, а також для аналізу сумісності додатків із апаратним забезпеченням [4].

Метою дослідження є вдосконалення процесу архітектурного вибору серверного середовища для телеметрії відеоігор шляхом розробки алгоритмічного методу аналізу впливу контейнеризації на швидкодію, що дозволить мінімізувати мережеві затримки та забезпечити стабільну роботу системи в умовах горизонтального масштабування. Гіпотеза дослідження вбачається у тому, що збільшення затримок запитів сервера телеметрії в контейнеризованому середовищі спричиняється накладними витратами мережевого стеку Docker Bridge, зокрема, ресурсомісткістю процесів NetFilter/IPTables під час обробки інтенсивних потоків запитів.

Для проведення дослідження поставлено такі задачі:

- розробка методу аналізу впливу контейнеризації на швидкодію;
- підготовка хмарного стенду;
- написання інструкцій для створення скриптів бенчмаркінгу;
- проведення бенчмаркінгу.

У процесі виконання завдань виділено ключові точки навантаження та визначена архітектура навантаження. Також здійснено контроль якості, валідацію логіки роботи скриптів та усунення логічних помилок, які часто генерують LLM-моделі для створення коду. Для однієї телеметричної події вибрано розмір у межах 1–2 кілобайт, що є типовим для RPG-відеоігор. Для експерименту використано безкоштовні ресурси хмарних постачальників, тому масштаби дослідження звужено до меж ознайомих можливостей хмар. Обмеження експерименту полягають у використанні ядер AMD Zen 4, NVMe PCI Express v4 накопичувача, оперативної пам'яті з корекцією помилок 5600 мегатранзакцій на секунду та затримками CL 46-45-45. Допустимі затримки трьох цільових процесів (50, 95 та 99) оцінки тривалості запитів не повинні перевищувати 50 мілісекунд.

Аналіз останніх досягнень та публікацій

Розробники відеоігор дедалі частіше використовують телеметрію в офлайн-іграх для дослідження балансу складності та геймплею в грі. Також дані телеметрії можуть використовуватися для розуміння переваг аудиторії [5], [6]. Для збору таких даних телеметрії потрібен серверний додаток, який має характерну для RPG-ігор структуру бази даних та може забезпечити надходження даних у режимі реального часу з низькими затримками. Серед наукових досліджень тема проектування архітектури бекенду ігрової телеметрії явно не висвітлена, проте існують реалізації бекенду ігрової телеметрії для інших інтерактивних систем, де авторами пропонується застосування семантичних правил під час прийому телеметрії в режимі реального часу [7]. Для розв'язання цієї задачі пропонується ідея застосування семантичного ядра, що використовує семантичні правила під час прийому телеметрії для стандартизації, валідації та надійного моніторингу інфраструктури. В межах дослідження офлайн RPG-ігор аналіз даних є окремим процесом, який відбувається паралельно до збору даних, тому потрібною є розробка архітектури ігрової телеметрії.

Сервери для збору телеметрії зазвичай розгортаються у хмарних датацентрах [8]. Серед досліджень, де розглядається продуктивність, масштабованість та затримки хмарних мережевих компонентів, що використовуються для забезпечення мережевого зв'язку в контейнерних середовищах у хмарах, виділимо статтю [9]. Автори провели серію досліджень для кількісної оцінки пропускну здатності протоколів TCP та UDP як у межах одного вузла, так і в багатовузлових конфігураціях хмарних систем [9]. У статті [10] досліджено можливості використання контейнеризації Docker та Kubernetes для спрощення, прискорення та масштабування процесу розгортання програм у високопродуктивних обчисленнях. Автори порівнюють ефективність, ресурсні витрати й безпеку контейнеризації з традиційним віртуалізованим методом розгортання мікросервісів [10].

Методика експерименту

Для подальших досліджень розроблено сервер збору телеметрії для аналізу аналітики. Важливо розуміти, що під час відлагодження ігрових додатків на сервер може надходити потік телеметрії високої щільності, тому під час його проектування необхідно врахувати як можливість отримання даних від різних клієнтів, так і можливість розгортання у різних серверних середовищах. Також під час проектування важливо передбачити можливість масштабування сервера та бази даних. Зручним є варіант масштабування за допомогою контейнеризації, зокрема, Docker та Kubernetes. З їх допомогою можна зібрати образ контейнера з API-сервером та забезпечити його горизонтальне масшта-

бування. Проте контейнеризація, як і віртуалізація, хоча і меншою мірою, створює накладні витрати системних ресурсів та швидкодії [11]. Саме тому для підвищення точності подальших досліджень з використанням горизонтального масштабування API-сервера збору телеметрії необхідно визначити вплив контейнеризації на швидкодію.

У реальних умовах експлуатації офлайн RPG-ігор клієнтські додатки не завжди з'єднані з мережею, наприклад, на ігрових консолях. Діагностичні дані, ігрова статистика та логи подій (наприклад, прогрес квестів чи отримання досягнень) акумулюються локально на пристрої гравця. Передача цих даних на хмарний сервер має залповий характер і активується автоматично за певними системними тригерами: у момент відновлення інтернет-з'єднання, по завершенню ігрової сесії або під час переходу між локаціями.

Метод аналізу впливу контейнеризації на швидкодію описано у вигляді алгоритму, показаному на рис. 1.

Алгоритм: Аналіз та вибір архітектури розгортання

Вхідні дані:

$C = \{VM1, VM2, VM3, VM4, VM5\}$ // Множина конфігурацій
 $S = \{Auth, Read, Write\}$ // Множина сценаріїв навантаження
 $th_lat = 50ms$ // Порогове значення для P99
 $th_p = 0.05$ // Рівень значущості p-value
 $th_d = 0.2$ // Поріг ефекту Коена

Вихідні дані:

a_opt // Рекомендована архітектура

FOR EACH c IN C :

DeployEnvironment(c) // Розгортання віртуальної машини
 WarmUp(c) // Прогрів кешу та бази даних
 ExecutionResult(c) = Benchmark(c , S) // Виконання тестів
 CollectMetrics(c) = {P50, P95, P99, Mean, StdDev}
 Shutdown(c) // Зупинка віртуальної машини

FOR EACH pair (c_i , c_j) IN C :

$p_value = WelchTTest(Results(c_i), Results(c_j))$
 $d_effect = CohenD(Results(c_i), Results(c_j))$
 IF ($p_value < th_p$ AND $d_effect > th_d$):
 LogDifference(c_i , c_j)

$filter_candidates = SelectWhere(P99 < th_lat)$

$A_opt = SortByMinP99(filter_candidates)$

RETURN a_opt

Рис. 1. Алгоритм роботи методу аналізу впливу контейнеризації на швидкодію

Визначено функції для псевдокоду алгоритму:

WarmUp(c) — етап попереднього виконання навантаження для виведення системи у стан стабільної продуктивності;

WelchTTest() — статистична перевірка різниці середніх значень затримок, що враховує потенційну нерівність дисперсій;

Cohen_D — оцінка практичної значущості різниці, що дозволяє відрізнити статистичну шумову різницю від архітектурно значущої;

SelectWhere() — фільтрація конфігурацій, що відповідають очікуваній затримці;

p_value — статистичний показник, який дозволяє оцінити, чи є отримані результати експерименту системними, чи вони могли бути спричиненими випадковими коливаннями.

Метод аналізу впливу контейнеризації на швидкодію реалізує послідовне виконання базового функціоналу:

1. Підготовка хмарної інфраструктури, яка полягає у розгортанні п'яти ідентичних віртуальних машин (ВМ) на базі гіпервізору VMWare ESXi з операційною системою Ubuntu 24.04 LTS та зарезервованих ресурсів, які функціонують у режимі послідовного запуску ВМ.
2. Конфігурація середовищ розгортання передбачає реалізацію п'яти варіантів поєднання нативного та Docker-контейнеризованого розгортання API-сервера та бази даних MySQL у різних конфігураціях, заданих табл. 1.
3. Розробка інструментарію бенчмаркінгу полягає у створенні скриптів для трьох сценаріїв навантаження (автентифікації, зчитування та запису даних) з обов'язковою ручною валідацією логіки для усунення помилок генерації коду.
4. Виконання бенчмаркінгу складається із запуску підготовлених скриптів на кожній віртуальній машині з використанням паралельних потоків у кожному варіанті для збору даних із фіксацією тривалості за 50-м, 95-м та 99-м процентилями.
5. Статистичний аналіз результатів базується на порівнянні показників затримок між конфігураціями із застосуванням t-тесту Велча [12] для перевірки значущості різниць та критерію ефекту Коена [13] для оцінки впливу архітектурних особливостей на продуктивність системи.

Таблиця 1

Конфігурація розгортань віртуальних машин

Віртуальна машина	Розгортання API	Розгортання БД	Мережа
VM1	нативне	нативне	мережа хосту
VM2	Docker	нативне	міст Docker, мережа хосту
VM3	нативне	Docker	мережа хосту, міст Docker
VM4	Docker	Docker	міст Docker
VM5	Docker	Docker	мережа хосту

Для проведення експерименту віртуальні машини розгорнуто на сервері з чистою інсталяцією гіпервізору VMWare ESXi. Віртуальні машини розгорнуто з ідентичних образів Ubuntu 24.04 LTS з повним резервуванням обчислювальних ресурсів для кожної: 2 vCPU та 4 GB RAM. Під час проведення бенчмаркінгу встановлено, що лише одна віртуальна машина може працювати на сервері.



Рис. 2. Архітектура розгортання віртуальної машини

```
FROM node:22-slim
WORKDIR /app
COPY package*.json ./
RUN npm ci --omit=dev
COPY app/ ./app/
RUN useradd --uid 1001 --no-create-home appuser
RUN chown -R appuser:appuser /app
USER appuser
EXPOSE 3000
CMD ["node", "app/server.js"]
```

Рис. 3. Код сценарію Dockerfile

штучного інтелекту Claude Opus 4.6, за допомогою якого розроблено скрипт бенчмаркінгу для трьох сценаріїв усіх віртуальних машин: випробування автентифікації клієнта, зчитування даних клієнта та запису статистичних даних (рис. 4).

Скрипт передбачав бенчмаркінг кожної розгорнутої віртуальної машини за трьома сценаріями, у кожному з яких утворювалося по 10 потоків, які мали виконувати по 200 конкуруючих запитів.

З настанням черги роботи наступної віртуальної машини, попередня вимикається. На кожній віртуальній машині розгорнуто API-сервер та сервер бази даних MySQL відповідно до конфігурації, поданої у табл. 1. Архітектура розгортання компонентів всередині віртуальної машини показана на рис. 2. Відповідно компоненти А чи Б можуть бути розгорнуті або безпосередньо у віртуальній машині, або у вигляді Docker контейнеру.

У нативних розгортаннях пакети node.js чи/та MySQL встановлювалися з deb-репозиторів Ubuntu GNU/Linux. У випадку розгортання Docker вибрано офіційні образи. Для розгортання API-сервера у середовищі Docker створено Dockerfile (рис. 3).

Для розгортання обох компонентів у середовищі Docker використано утиліту docker-compose. Для подальшого випробування створено документаційні файли та запит до LLM-моделі

Такий вибір кількості запитів здійснено для репрезентативності пікових навантажень 2000 одночасних з'єднань, що є типовим для RPG-відеоігор. За такого підходу результати експерименту будуть концентруватися не на аналізі динамічного навантаження на сервер, а саме на граничних випадках максимального навантаження.

1. Analyze a project info at files @README.md and @package.json
2. Analyze server's endpoints documentation from @doc/openapi.yaml
3. Read @doc/ANALYTICS.md with guide of percentile latency metrics calculation
4. Read @doc/deployments/VM.md with configurations of each VM
5. Read @doc/deployments/ssh.conf with access configuration for each VM via SSH
6. Create a node.js script at @scripts/benchmark.js providing following evaluation scenarios:
 - Scenario 1: Auth Layer - @tests/scenario1.txt
 - Scenario 2: Player Upsert - @tests/scenario2.txt
 - Scenario 3: Stats Write - @tests/scenario3.txt

Рис. 4. Інструкції для LLM-моделі Claude Opus 4.6 для створення скрипта бенчмаркінгу

Інструкції передбачали розбиття тривалості запитів за трьома процентилями: 50, 95 та 99. Скрипт з авторськими змінами завантажено у репозиторій Github [14].

Проведення експерименту передбачає порівняння за допомогою t-тесту Велча для перевірки статистичної значущості малих різниць у затримках мережі між конфігураціями віртуальних машин.

Результати дослідження

Перед початком випробування користувача зареєстровано на всіх віртуальних машинах. Скрипти бенчмаркінгу запущено стосовно всіх VM. Авторизовані користувачі на API-сервері зберігаються в оперативній пам'яті, через що перевірка авторизації є хорошим способом бенчмаркінгу API-сервером за запитами, які не потребують окремих запитів до бази даних. Результати першого бенчмаркінгу за допомогою запитів авторизації відображені у табл. 2, де σ є вибірковою стандартною відхиленням затримок для проведених 2000 запитів навантаження.

Таблиця 2

Бенчмаркінг за сценарієм № 1

Віртуальна машина	П. 50 (мс)	П. 95 (мс)	П. 99 (мс)	Середні (мс)	σ (мс)	Запитів/сек
VM1	12,917	14,004	15,457	13,017	0,609	728,56
VM2	12,981	14,067	15,560	13,061	0,593	725,48
VM3	12,975	14,886	18,845	13,227	1,179	715,91
VM4	13,069	14,174	15,253	13,158	0,611	722,24
VM5	12,998	25,693	33,419	14,283	4,211	658,20

Другий сценарій бенчмаркінгу передбачав зчитування даних про клієнта з бази даних сервера телеметрії. Сервер телеметрії виконував запити SELECT до бази даних. Результати зведено у табл. 3. Збільшення часу виконання процесів обумовлено тим, що запит SELECT потребує більше часу на виконання, ніж зчитування даних з оперативної пам'яті.

Таблиця 3

Бенчмаркінг за сценарієм № 2

Віртуальна машина	П. 50 (мс)	П. 95 (мс)	П. 99 (мс)	Середні (мс)	σ (мс)	Запитів/сек
VM1	13,081	14,010	16,042	13,185	0,723	727,39
VM2	13,209	14,426	16,164	13,340	0,692	719,01
VM3	13,153	14,693	17,306	13,345	0,892	716,36
VM4	13,203	14,258	15,171	13,289	0,534	721,40
VM5	13,153	14,220	15,829	13,256	0,578	722,14

Третій сценарій бенчмаркінгу передбачав запис статистичних даних телеметрії у базу даних (табл. 4). Особливістю цього процесу є використання операції INSERT. Цілком очікувано, що операції запису займатимуть більше часу, ніж операції зчитування, оскільки результат SQL-запитів

INSERT та UPDATE може бути повернутим до фактичного запису даних на диск [15].

Таблиця 4

Бенчмаркінг за сценарієм № 3

Віртуальна машина	П. 50 (мс)	П. 95 (мс)	П. 99 (мс)	Середні (мс)	σ (мс)	Запитів/сек
VM1	13,282	14,806	17,995	13,482	0,986	710,06
VM2	13,495	15,255	17,014	13,691	0,887	700,23
VM3	13,449	14,922	16,052	13,603	0,718	702,21
VM4	13,491	15,182	20,349	13,747	1,454	697,80
VM5	13,464	14,788	15,948	13,599	0,686	704,47

Віртуальна машина VM4 має найвище середнє значення, найбільше вибіркове стандартне відхилення (σ) та найвищий П.99 у сценарії 3 — робочому навантаженні з перевагою операцій запису, де накладні витрати мережевого стеку Docker є найпомітнішими. У першому сценарії бенчмаркінгу результати віртуальної машини VM5 становлять виняток за затримками. Коли два Docker-контейнери запускаються у мережевому просторі хосту, сервіс Docker підтримує правила IPTables на рівні хостової операційної системи для обліку ізоляції мережі та маскування будь-якого вихідного трафіку [11]. Припущено, що під паралельним навантаженням очищення та перебудова таблиць правил IPTables на коротку мить посилюють конкуренцію за блокування IPTables на хості [9]. Це призводить до різкого стрибка затримки під час TCP-рукоштовування або отримання першого байта для тих запитів, які випадково надійшли у вікно такого очищення. Порівняння показників віртуальних машин зведено у табл. 5.

Таблиця 5

Порівняння показників віртуальних машин

Порівняння віртуальних машин		Сценарій № 2			Сценарій № 3		
		$\Delta_{сер}$ (мс)	t	d	$\Delta_{сер}$ (мс)	t	d
VM1	VM2	0,155	6,93	0,219	0,209	7,05	0,223
VM1	VM3	0,160	6,23	0,197	0,121	4,44	0,140
VM5	VM4	0,033	1,88	0,059	0,148	4,12	0,130
VM1	VM5	0,071	3,43	0,109	0,117	4,36	0,138

У табл. 5 параметр $\Delta_{сер}$ — це різниця середніх значень затримки між двома віртуальними машинами у мілісекундах, t — статистичний показник, який використовується для перевірки, чи є виявлена різниця між середніми значеннями двох тестів статистично значущою, чи це випадковість. Якщо t -тест підтверджує наявність різниці, то d Коена допомагає зрозуміти, наскільки ця різниця є суттєвою. Для значень t вважається, що коли $t > 3,29$, то результат є достовірним на рівні значущості 0,001, коли $t > 2,58$ — достовірним на рівні значущості 0,01, коли $t > 1,96$ — достовірним на рівні значущості 0,05, коли $t \leq 1,96$ — статистично незначимим. Для значень $d < 0,2$ вважається, що різниця є несуттєвою, а значення 0,2...0,5 вказують на наявність малого ефекту.

Проведений статистичний аналіз підтверджує значущість втрат на обробку мережевого трафіку для конкретного контрольованого стенду. Зокрема, у порівнянні віртуальних машин VM4 та VM5 різниця середніх затримок становила 0,148 мс ($t = 4,12$, $d = 0,130$), а для пари віртуальних машин VM1 та VM2 цей показник досяг 0,209 мс ($t = 7,05$; $d = 0,223$). Так статистичний аналіз за t -тестом Велча довів, що ізолюваний вплив NetFilter (віртуальної машини VM4 проти VM5) стає статистично значущим за умов інтенсивного навантаження на запис, де накладні витрати контейнеризації і мережевого стеку NetFilter рівнозначні за впливом на загальну затримку. Водночас ключова вразливість конфігурації Docker Bridge полягає не у сталому проходженні трафіку, а в деградації стабільності системи, спричиненій конкуренцією за блокування IPTables під час внутрішніх фонових процесів сервісу Docker. Для систем з критичними вимогами до стабільної затримки, такими як сервер телеметрії для офлайн RPG-відеоігор, та з великою кількістю паралельних запитів рекомендується використовувати режим мережевого простору хосту, тоді як для стандартних операцій читання архітектура Docker Bridge є цілком виправданою та ефективною.

Висновки

Розроблений метод аналізу впливу контейнеризації на швидкодію ґрунтується на багатofакторному порівняльному аналізі з використанням методів статистичної перевірки за t -тестом Велча та критерієм ефекту Коена. Запропонований метод дозволяє з високою точністю диференціювати випадкові шуми вимірювань від системного впливу контейнеризації на продуктивність роботи сервера телеметрії, забезпечуючи обґрунтованість вибору оптимальної архітектури для систем з критичними вимогами до затримки в офлайн RPG-іграх.

Підготовка хмарного стенду полягала у розгортанні п'яти віртуальних машин на базі гіпервізору VMware ESXi, кожна з яких використовувала ідентичні образи ОС Ubuntu 24.04 LTS із зарезервованими обчислювальними ресурсами. В межах експерименту на цих машинах реалізовано різні комбінації нативного та контейнеризованого розгортання API-сервера та бази даних MySQL, що дозволило створити середовище для порівняльного аналізу продуктивності залежно від архітектурних особливостей.

Процес написання інструкцій для створення скриптів бенчмаркінгу дозволив формалізувати вимоги до генерації програмного коду за допомогою LLM-моделі Claude Opus 4.6, забезпечивши бенчмаркінг для трьох ключових сценаріїв: автентифікації, зчитування та запису статистичних даних. Важливим етапом став контроль якості та ручна валідація логіки скриптів, що мінімізувало ризики, пов'язані з помилками LLM-генерації, та дозволило отримати високоефективний інструментарій.

За результатами проведеного експерименту встановлено, що архітектура Docker Bridge зумовлює додаткові накладні витрати на обробку мережевого трафіку, які стають особливо помітними під час інтенсивних операцій запису даних. Виявлена нестабільність, спричинена конкуренцією за ресурси мережевого стеку, підтверджує гіпотезу та дозволяє рекомендувати використання режиму мережевого простору хосту для систем з критичними вимогами до стабільності затримок, тоді як для стандартних завдань зчитування даних використання контейнеризації є досить ефективним рішенням.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

- [1] O. V. Talaver, and T. A. Vakaliuk, "Dynamic system analysis using telemetry," in *CS & SE @ SW*, pp. 193-209, 2023. [Electronic resource]. Available: <https://ceur-ws.org/Vol-3662/paper01.pdf>.
- [2] А. С. Авраменко, В. С. Авраменко, і Г. В. Косенюк, *Тестування програмного забезпечення*, навч. посіб. Черкаси: ЧНУ імені Богдана Хмельницького, 2017. [Електронний ресурс]. Режим доступу: <https://eprints.cdu.edu.ua/1482/>.
- [3] L. Franke, H. Liang, S. Farzanehpour, A. Brantly, J. C. Davis, and C. Brown, "An exploratory mixed-methods study on General Data Protection Regulation (GDPR) compliance in open-source software," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Oct. 2024, pp. 325-336. <https://doi.org/10.1145/3674805.3686692>.
- [4] J. Riedesel, *Software Telemetry: Reliable Logging and Monitoring*. Shelter Island, NY, USA: Manning Publications, 2021. [Electronic resource]. Available: <https://www.oreilly.com/library/view/software-telemetry/9781617298141/>.
- [5] A. Drachen, "What is game telemetry?," *GameAnalytics*, May 16, 2025. [Electronic resource]. Available: <https://www.gameanalytics.com/blog/what-is-game-telemetry>. Accessed: 13.05.2026.
- [6] П. Малініч, і В. Войтко, «Програмна архітектура взаємозв'язку аналітики телеметрії відеоігор та динамічної генерації ігрового контенту,» *Herald of Khmelnytskyi National University. Technical Sciences*, vol. 355, no. 4, pp. 365-372, Aug. 2025. <https://doi.org/10.31891/2307-5732-2025-355-52>.
- [7] O. Osolinskiy, K. Lipianina-Honcharenko, and M. Komar, "Semantic Core for Sensor Telemetry Ingestion for Digital Twins," *Smart Cities*, vol. 9, no. 5, p. 77, 2026. <https://doi.org/10.3390/smartcities905007>.
- [8] A. Al Maruf, A. Bakhtin, T. Cerny, and D. Taibi, "Using microservice telemetry data for system dynamic analysis," in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, Aug. 2022, pp. 29-38, <https://doi.org/10.48550/arXiv.2207.02776>.
- [9] J. Claassen, R. Koning, and P. Grosso, "Linux containers networking: Performance and scalability of kernel modules," in *NOMS 2016. 2016 IEEE/IFIP Network Operations and Management Symposium*, Istanbul, Turkey, 2016, pp. 713-717. <https://doi.org/10.1109/NOMS.2016.7502883>.
- [10] Г. Г. Киричек, В. В. Смірнов, і М. Ю. Тягунова, «Дослідження застосування контейнерних технологій для розгортання програм на суперкомп'ютерах,» *Вісник Кременчуцького національного університету імені Михайла Остроградського*, № 3, 2023. <https://doi.org/10.32782/1995-0519.2023.3.4>.
- [11] І. П. Малініч, і В. І. Месюра, «Ін'єктивний метод отримання даних користувацького досвіду в ігрових симуляторах комп'ютерних мереж,» *Вісник Вінницького політехнічного інституту*, № 5, с. 49-54, 2019. <https://doi.org/10.31649/1997-9266-2019-146-5-49-54>.
- [12] M. Bland, *An introduction to medical statistics*. Oxford, UK: Oxford University Press, 2015. [Online]. Available: <https://global.oup.com/academic/product/an-introduction-to-medical-statistics-9780199589920>.
- [13] Z. Zhang, and L. Wang, *Advanced statistics using R*. Granger, in: ISDSA Press, 2017. <https://doi.org/10.35566/advstats>.
- [14] *Telemetry Benchmark – GitHub* [Electronic resource]. Available: <https://github.com/linuxation/telemetry-benchmark>. Accessed: 13.06.2026.
- [15] О. Н. Романюк, і Т. О. Савчук, *Організація баз даних і знань*, навч. посіб. Вінниця: УНІВЕРСУМ-Вінниця, 2003. [Електронний ресурс]. Режим доступу: <https://ir.lib.vntu.edu.ua/handle/123456789/19527>.

Рекомендована кафедрою програмного забезпечення ВНТУ

Дата надходження 25.05.2026

Дата прийняття після доопрацювання 30.06.2026

Дата прийняття до друку після рецензування 2.07.2026

Дата публікації 2.10.2026

Малініч Павло Павлович — аспірант кафедри програмного забезпечення, e-mail: pavlo.malinich@vntu.edu.ua . <https://orcid.org/0009-0008-2423-9100>;

Войтко Вікторія Володимирівна — канд. техн. наук, доцент, доцент кафедри програмного забезпечення, e-mail: dekanfki@i.ua . <https://orcid.org/0000-0002-3329-7256>.

Вінницький національний технічний університет, Вінниця

P. P. Malinich¹

V. V. Voitko¹

Research of Telemetry API Server Latency for Offline RPG Video Games

¹Vinnytsia National Technical University

Telemetry is an important element of the modern software support process, involving the collection of diagnostic data for the prompt resolution of technical defects. A classic example of telemetry usage is the submission of critical error reports: in the event of an application crash, the system automatically transmits a memory dump to developers to analyze the root cause of the failure. In scenarios where a program operates without continuous network access, analytics are accumulated locally. The transmission of this data to servers is triggered automatically as soon as the internet connection is restored or when specific system events occur. Video game publishers, particularly of offline RPGs, actively leverage telemetry to analyze difficulty balance, evaluate gameplay, and understand their audience's preferences. This requires a server application deployed in cloud data centers, featuring a database structure typical of RPGs, and capable of ensuring real-time data ingestion with low latency. As part of the study, an experimental testbed was deployed using five virtual machines running the Ubuntu GNU/Linux OS on a VMware ESXi hypervisor. A comparative analysis was conducted on five architectural configurations that combine native deployment and Docker containerization for the API server and MySQL database. The methodology relies on load testing scenarios that simulate the peak burst workloads characteristic of game clients reconnecting to the network. The results figured out that the most efficient architecture for a game telemetry server utilizes host networking, whereas standard read and authorization operations are successfully executed within isolated container networks. The developed method enables the selection of an optimal server environment, minimizing network latency and ensuring infrastructure scalability to support a large number of concurrent connections.

Keywords: telemetry, video games, latency, server, cloud, GNU/Linux, Docker.

Malinich Pavlo P. — Post-Graduate Student of the Chair of Software Development, e-mail: pavlo.malinich@vntu.edu.ua . <https://orcid.org/0009-0008-2423-9100>;

Voitko Viktoriia V. — Cand. Sc. (Eng.), Associate Professor, Associate Professor of the Chair of Software Development, e-mail: dekanfki@i.ua . <https://orcid.org/0000-0002-3329-7256>