УДК 004.65:004.77:658.7 DOI https://doi.org/10.35546/kntu2078-4481.2025.4.3.22

Д. М. РАТУШНИЙ
аспірант кафедри комп'ютерних наук
Національний університет біоресурсів та природокористування України
d.ratushny@nubip.edu.ua
ORCID: 0009-0009-4699-9909

Р. А. РУДЕНСЬКИЙ
доктор економічних наук, професор,
професор кафедри компьютерних наук
Національний університет біоресурсів та природокористування України
ORCID: 0009-0002-3682-9702

АРХІТЕКТУРНІ ПІДХОДИ ДО ВИБОРУ СТЕКУ ТЕХНОЛОГІЙ ДЛЯ ЗБЕРІГАННЯ IOT-ДАНИХ В ЗАДАЧАХ УПРАВЛІННЯ ЛОГІСТИКОЮ

Стрімкий розвиток концепції Логістика 4.0 (Logistics 4.0) та впровадження технологій Інтернету речей (IoT) у ланцюги постачання призвели до експоненційного зростання обсягів телеметричних даних. Сучасні інформаційні системи управління логістикою вимагають обробки потоків даних від GPS-трекерів, датчиків температури та сенсорів рівня пального в режимі реального часу. Специфіка цих даних, що відносяться до категорії часових рядів (Time-Series Data), створює суттєве навантаження на підсистеми зберігання, з яким традиційні реляційні системи управління базами даних (РСУБД) часто не справляються ефективно. У даній статті проведено комплексне дослідження архітектурних підходів до побудови підсистеми зберігання та обробки IoT-даних. Метою роботи є визначення оптимального технологічного стеку, здатного забезпечити як високу швидкість запису даних (ingestion rate), так і низьку затримку при виконанні аналітичних запитів, що є критичним для подальшого застосування методів машинного навчання та технології блокчейн. Для досягнення мети розроблено експериментальний стенд на базі контейнеризації Docker, що імітує навантаження від автопарку (до 1000 вантажівок) із генерацією даних у режимі реального часу. Проведено порівняльний аналіз продуктивності чотирьох класів СУБД: реляційної (PostgreSQL), гібридної (TimescaleDB), спеціалізованої NoSQL (InfluxDB) та In-Memory (Redis). Встановлено, що InfluxDB демонструє найвищу швидкість запису (~121 800 повідомлень/сек), випереджаючи PostgreSQL майже втричі. Водночас, для задач оперативного моніторингу лідером визначено Redis із часом відгуку менше 1 мс. Виявлено, що TimescaleDB, маючи нижчу швидкість запису (~14 500 повідомлень/сек) через витрати на партиціонування, забезпечує компроміс між продуктивністю та SQL-аналітикою. На основі отриманих даних запропоновано гібридну архітектуру: Redis як "гарячий" шар для диспетчеризації та TimescaleDB/InfluxDB як "холодний" шар для ML-моделювання та аудиту.

Ключові слова: *Інтернет речей, IoT, бази даних часових рядів, Time-Series Database, логістика, InfluxDB, TimescaleDB, Redis, PostgreSQL, високонавантажені системи.*

D. M. RATUSHNYI
Postgraduate Student at the Department of Computer Sciences
National University of Life and Environmental Sciences of Ukraine
ORCID: 0009-0009-4699-9909

R. A. RUDENSKYI
Doctor of Economic Sciences, Professor,
Professor at the Department of Computer Sciences
National University of Life and Environmental Sciences of Ukraine
ORCID: 0009-0002-3682-9702

ARCHITECTURAL APPROACHES TO SELECTING A TECHNOLOGY STACK FOR IOT DATA STORAGE IN LOGISTICS MANAGEMENT TASKS

The rapid development of the Logistics 4.0 concept and the widespread implementation of Internet of Things (IoT) technologies in supply chains have led to an exponential increase in telemetry data volumes. Modern logistics management information systems require processing data streams from GPS trackers, temperature sensors, and fuel level sensors in

real-time. The specific nature of this data, which belongs to the Time-Series category, creates a significant load on storage subsystems, which traditional relational database management systems (RDBMS) often fail to handle effectively. The purpose of this paper is to determine the optimal technology stack capable of ensuring both high data ingestion rates and low latency when executing analytical queries, which is critical for the further application of machine learning methods and blockchain technology.

Methodology. To achieve this goal, an experimental testbed based on Docker containerization was developed, simulating the load from a fleet of vehicles (up to 1000 trucks) with real-time data generation. A comparative performance analysis of four classes of DBMS was conducted: relational (PostgreSQL), hybrid (TimescaleDB), specialized NoSQL (InfluxDB), and In-Memory (Redis). The study involved two stages: an Ingestion Benchmark (measuring write throughput) and a Query Benchmark (measuring latency for operational and analytical queries).

Findings. It was established that InfluxDB demonstrates the highest write speed (~121,800 messages/sec), outperforming PostgreSQL by almost three times. This confirms the efficiency of LSM-tree structures for write-heavy workloads. At the same time, Redis was identified as the leader for operational monitoring tasks with a response time of less than 1 ms. It was revealed that TimescaleDB, having a lower write speed (~14,500 messages/sec) due to partitioning overheads, provides a compromise between performance and SQL analytics capabilities.

Scientific Novelty. The limits of applicability for hybrid SQL-based time-series systems in logistics have been experimentally defined. It is proven that for loads exceeding 15,000 events per second, specialized NoSQL solutions are required. An architectural limitation of Redis in scenarios of massive writing of heterogeneous metrics due to its single-threaded model was identified.

Practical Value. Based on the obtained data, a hybrid two-tier architecture is proposed: Redis as a "hot" layer for dispatching and real-time tracking, and TimescaleDB/InfluxDB as a "cold" layer for deep analytics, ML modeling, and audit trails. This approach allows leveling the shortcomings of each individual technology.

Key words: *Internet of Things, IoT, Time-Series Database, logistics, InfluxDB, TimescaleDB, Redis, PostgreSQL, high-load systems.*

Постановка проблеми

Сучасна логістика переживає фундаментальну трансформацію, переходячи від реактивних моделей управління до проактивних, що базуються на даних. Ключовим драйвером цих змін є масове впровадження IoT-пристроїв. Транспортні засоби та контейнери оснащуються сенсорами, які безперервно генерують потоки даних про геолокацію, температурний режим та технічний стан.

Ці дані мають природу часових рядів (Time-Series Data): вони є незмінними, прив’язані до часу і надходять з високою частотою. Ефективне управління ними є фундаментом для побудови систем Логістики 4.0, що використовують машинне навчання для прогнозування та блокчейн для аудиту. Проте стандартні реляційні бази даних часто стикаються з проблемами масштабування при спробі обробити тисячі запитів на запис в секунду.

Аналіз останніх досліджень і публікацій

Стрімке зростання кількості IoT-пристроїв призводить до генерування величезних обсягів даних. Зокрема, прогнозується, що до 2020 року кількість підключених датчиків сягне близько 24 млрд одиниць [1]. Дані, що збираються в таких системах, характеризуються високою частотою надходження та значними обсягами [2–3], тобто належать до категорії *Big Data*. Відповідно, ефективне збереження й обробка IoT-телеметрії стають серйозним викликом. Традиційні реляційні СУБД виявляються недостатньо продуктивними за подібних навантажень, особливо коли йдеться про тисячі операцій запису за секунду [4–6]. На думку низки дослідників, для таких випадків краще підходять нереляційні NoSQL-системи, які спроектовані з урахуванням масштабованості та розподіленої обробки даних IoT [4–6]. Окрім NoSQL, останніми роками з’явився окремий клас спеціалізованих *time-series* СУБД, оптимізованих під роботу з часовими рядами (InfluxDB, OpenTSDB, TimescaleDB тощо). Проведено кілька оглядів та порівнянь можливостей таких систем. Зокрема, у роботі [7] наведено огляд відкритих *time-series* баз даних і показано їх переваги при обробці послідовних даних. Для збереження сумісності з SQL-екосистемою розроблено гібридні рішення, такі як TimescaleDB – розширення PostgreSQL, що автоматично партиціонує часові ряди. За рахунок цього TimescaleDB забезпечує значно вищу швидкість вставки даних та виконання часових запитів порівняно зі звичайною PostgreSQL [8]. Низка дослідників зосередилася на порівняльному аналізі продуктивності різних підходів до зберігання IoT-даних. Так, у роботі [9] проаналізовано ефективність кількох СУБД при збереженні даних моніторингу якості води (показано, зокрема, переваги InfluxDB над реляційним рішенням для великої кількості сенсорів). Інше дослідження [10] присвячене оцінюванню TimescaleDB, InfluxDB, Riak TS та SQLite на низькопотужному пристрої Raspberry Pi; його результати продемонстрували, що за певних умов навіть традиційна PostgreSQL може перевершувати спеціалізовані *time-series* СУБД за швидкістю запису даних [10]. Окрім вибору типу СУБД, увага приділяється й оптимізації налаштувань баз даних під IoT-навантаження. Зокрема, в [11] досліджено вплив стратегії компакції в Cassandra на продуктивність: показано, що мінімальне вікно компакції (1 хв) забезпечує найвищий темп запису, проте потребує приблизно на 20% більше дискового простору порівняно з більшими інтервалами [11]. Таким чином, аналіз останніх досліджень показує, що хоча існує багато окремих рішень для зберігання часових рядів та великих даних IoT, досі відсутнє комплексне порівняння

представників різних класів СУБД (реляційних, розширених SQL, NoSQL та *in-memory*) в умовах реальних логістичних IoT-навантажень. Це зумовлює актуальність даного дослідження та визначає його мету.

Формулювання мети дослідження

Мета статті – обґрунтування вибору оптимального архітектурного стеку технологій для зберігання IoT-даних у логістичних системах шляхом експериментального порівняння чотирьох підходів: реляційного (PostgreSQL), розширеного SQL (TimescaleDB), нативного NoSQL (InfluxDB) та In-Memory (Redis).

Викладення основного матеріалу дослідження

Архітектура експериментального стенду. Для забезпечення ізоляції процесів та відтворюваності результатів (reproducibility), експериментальне середовище було розгорнуто з використанням технології контейнеризації Docker. Всі сервіси функціонували в межах єдиної віртуальної мережі (Docker Network bridge mode), що дозволило мінімізувати мережеві затримки, пов'язані з маршрутизацією трафіку між різними хостами, і зосередитися на вимірюванні продуктивності самих СУБД.

Апаратна конфігурація тестового вузла:

- **Процесор (CPU):** Intel Core i5-11400H (6 ядер, 12 потоків, тактова частота до 4.5 ГГц);
- **Оперативна пам'ять (RAM):** 16 ГБ DDR4 3200 МГц;
- **Дискова підсистема:** NVMe SSD M.2 (швидкість послідовного запису до 2500 МБ/с);
- **Операційна система:** Ubuntu Linux 22.04 LTS (Kernel 5.15).

Програмне забезпечення було налаштовано через оркестратор docker-compose. Для кожного контейнера було виділено ідентичні ліміти ресурсів (без жорстких обмежень CPU, для емуляції поведінки системи при пікових навантаженнях).

Характеристика досліджуваних систем зберігання даних

У дослідженні порівнюються чотири принципово різні підходи до організації зберігання часових рядів. Детальні параметри конфігурації наведено в Таблиці 1.

1. PostgreSQL 16 (Relational Baseline). Використовується як контрольна група. Дані зберігаються у "пласкій" таблиці (Heap Table). Для прискорення вибірки створено композитний B-Tree індекс по полях (truck_id, timestamp DESC). Цей підхід є стандартом для більшості корпоративних систем, проте має відомі обмеження при зростанні обсягу індексу, який перестає вміщуватися в оперативну пам'ять (RAM).
2. TimescaleDB (SQL-based Time-Series). Це розширення (extension) для PostgreSQL, яке впроваджує концепцію "гіпертаблиць" (Hypertables). Гіпертаблиця – це абстракція, яка автоматично розбиває (partitioning) дані на менші фізичні таблиці ("чанки" або chunks) на основі часового інтервалу.

Налаштування: Розмір чанку (chunk_time_interval) встановлено на 24 години. Це критичний параметр, оскільки він визначає, чи вміститься активний індекс (hot index) у кеш процесора та RAM.

3. InfluxDB v2.7 (Native NoSQL Time-Series). Використовує спеціалізований рушій зберігання TSM (Time-Structured Merge Tree) та індекс TSI (Time Series Index). На відміну від B-Tree, TSM оптимізовано для інтенсивного запису (append-only) та високого ступеня стиснення даних (Delta-of-Delta encoding для часових міток, Gorilla compression для чисел з плаваючою комою.

Налаштування: Використано стандартний bucket без обмежень retention policy для чистоти експерименту.

4. Redis Stack + RedisTimeSeries (In-Memory). База даних типу "ключ-значення", що працює виключно в оперативній пам'яті. Модуль RedisTimeSeries додає структури даних для ефективного зберігання та агрегації часових рядів (з використанням Double-Delta компресії).

Стратегія запису: Для уникнення конкуренції за ресурси (lock contention) застосовано архітектуру "одна метрика – один ключ" (per-metric keys), де ідентифікатор ключа формується як metric:truck_id.

Таблиця 1

Конфігурація досліджуваних СУБД

Параметр	PostgreSQL	TimescaleDB	InfluxDB	Redis
Версія	16.1-alpine	latest-pg16	2.7.1	Stack 7.2
Модель даних	Таблична (Rows)	Гіпертаблиця (Chunks)	Line Protocol (Points)	TimeSeries Key
Індексація	B-Tree	B-Tree (на чанках)	TSI (Inverted Index)	Hash Slot
Протокол	Binary (TCP)	Binary (TCP)	HTTP REST (JSON/Text)	RESP (Binary)
Стиснення	TOAST (для тексту)	Columnar (опціонально)	Gorilla / Delta	Double-Delta

Джерело: власна розробка автора

Математична модель даних та генератор навантаження

Для емуляції потоку даних було розроблено симулятор на платформі **Node.js**, що використовує бібліотеки драйверів pg, @influxdata/influxdb-client та redis. Вибір Node.js зумовлений його асинхронною, подіє-орієнтованою архітектурою (Event Loop), що дозволяє генерувати тисячі запитів на секунду з одного потоку виконання.

Вхідний потік даних моделюється як множина кортежів D, де кожен елемент d представляє стан датчиків транспортного засобу в момент часу t. Формально структуру повідомлення можна представити виразом (1):

$$d_i = \{t, id, lat, lon, T, F\}$$

де:

- t – часова мітка (UNIX timestamp з точністю до мілісекунд);
- id – унікальний ідентифікатор пристрою;
- lat, lon – географічні координати ;
- T – температура вантажу, нормальний розподіл навколо 4°C;
- F – рівень пального.

Генератор навантаження працює в режимі "максимальної пропускної здатності" (Maximum Throughput). Алгоритм генерації реалізовано наступним чином:

1. Ініціалізація пулу з'єднань з БД.
2. Формування пакету даних (Batch) розміром B = 1000 записів. Для кожного запису випадковим чином обирається id та генеруються показники сенсорів.
3. Асинхронна відправка пакету в БД.
4. Повторення кроків 2-3 без штучних затримок (sleep) до завершення часу тестування (T = 30 с).

Такий підхід дозволяє виявити "вузьке місце" (bottleneck) саме на стороні бази даних, оскільки генератор завідомо здатен створити навантаження, що перевищує можливості запису дискової підсистеми.

Алгоритм проведення експериментів та метрики

Дослідження розділене на два незалежних етапи для оцінки різних аспектів продуктивності.

Етап 1. Бенчмарк запису (Ingestion Benchmark).

Критично важливий для IoT-систем, де втрата даних через переповнення буфера є неприпустимою.

- **Процедура:** Запуск симулятора в режимі ingest для кожної СУБД окремо. Перед кожним тестом база даних очищується (truncate/flush) для уникнення впливу попередніх даних.

Вимірювана метрика: Throughput (RPS) – кількість успішно збережених записів за секунду, який розраховується за формулою (2):

$$RPS = \frac{N_{total}}{T_{end} - T_{start}}$$

де:

- N total – загальна кількість записаних подій;
- T start – час початку та тестування (мс);
- T end – час завершення тестування (мс).

Етап 2. Бенчмарк читання (Query Benchmark).

Оцінює придатність системи для задач оперативного моніторингу та аналітики. Попередньо в кожну БД завантажується масив даних обсягом 1.5 * 10^6 записів.

Виконуються два типи запитів:

1. **Запит Q1 (Operational):** "Показати останні 10 точок треку для вантажівки X". Цей запит емулює роботу диспетчера, який спостерігає за рухом авто.
   - *SQL:* SELECT * FROM sensor_data WHERE truck_id = ? ORDER BY timestamp DESC LIMIT 10
   - *InfluxQL:* from(bucket)… filter(id)… sort(desc)… limit(10)
2. **Запит Q2 (Analytical):** "Розрахувати середню температуру за 24 години з агрегацією по 1 годині". Цей запит емулює роботу ML-алгоритму або BI-звіту.
   - *SQL:* SELECT time_bucket('1 hour', timestamp), AVG(temperature)… GROUP BY 1
   - *Redis:* TS.MRANGE … AGGREGATION AVG 3600000

- **Вимірювана метрика:** *Latency* – час від моменту відправки запиту до отримання повної відповіді клієнтом. Для нівелювання мережевих флуктуацій виконується 100 ітерацій кожного запиту, результат усереднюється.

Результати дослідження

Експериментальне дослідження проводилося у два етапи, що відповідають двом ключовим сценаріям використання баз даних у системах Логістики 4.0: інтенсивний прийом даних від сенсорів (Telemetry Ingestion) та виконання різнопланових запитів (Telemetry Querying).

Етап 1: Аналіз пропускної здатності запису (Ingestion Rate)

На цьому етапі емулювалася робота високонавантаженого шлюзу даних (IoT Gateway). Симулятор генерував потік повідомлень від 1000 унікальних ідентифікаторів (truck_id) із частотою, що обмежувалася лише швидкістю обробки транзакцій базою даних. Результати вимірювань, усереднені за 30 секунд тестування, наведено в Таблиці 2.

Таблиця 2

Результати навантажувального тестування (Ingestion Benchmark)

СУБД	Середній RPS (записів/сек)	Відносна продуктивність	Пікове споживання CPU
InfluxDB v2	121,800	100% (База)	85% (Multi-core)
PostgreSQL 16	43,800	35.9%	60%
Redis Stack	15,833	12.9%	98% (Single core)
TimescaleDB	14,567	11.9%	75%

Джерело: власна розробка автора

Аналіз результатів запису:

1. **Лідерство InfluxDB.** СУБД InfluxDB продемонструвала найкращий результат, обробляючи понад 120 тисяч метрик на секунду. Це пояснюється використанням LSM-дерева (Log-Structured Merge-tree) замість традиційного B-дерева. Дані спочатку потрапляють у буфер пам'яті (WAL + Cache), а потім послідовно скидаються на диск у вигляді незмінних файлів TSM (Time-Structured Merge), що мінімізує кількість операцій довільного доступу до диска (Random I/O.
2. **Феномен PostgreSQL.** "Чистий" PostgreSQL показав несподівано високий результат (43 800 RPS). Це зумовлено тим, що за відсутності складної логіки партиціонування, операція INSERT є простою процедурою додавання запису в кінець файлу кучі (Heap File) та оновлення одного B-Tree індексу. Однак, слід зазначити, що така продуктивність є тимчасовою: зі зростанням таблиці до розмірів, що перевищують обсяг RAM, швидкість вставки в B-Tree індекс деградуватиме логарифмічно.
3. **Обмеження Redis.** Результат Redis (15 833 RPS) виявився нижчим за очікуваний для In-Memory системи. Детальний аналіз показав, що вузьким місцем стала однопоточна архітектура Redis (Single-threaded Event Loop). При масовій вставці у тисячі різних ключів (стратегія Unique Key per Device) значна частина процесорного часу витрачалася на хешування ключів та управління словником даних, а не на саме збереження значень.
4. **Накладні витрати TimescaleDB.** Ця СУБД показала найнижчу швидкість запису, що є платою за автоматичне управління даними. При кожній вставці TimescaleDB виконує обчислення, щоб визначити, у який саме часовий сегмент ("чанк") має потрапити запис, і чи потрібно створювати новий чанк.

Етап 2: Аналіз латентності запитів (Query Latency)

На другому етапі база даних була попередньо наповнена масивом у 1.5 млн записів. Вимірювався час виконання ("round-trip time") для двох типів запитів. Результати наведено в Таблиці 3.

Таблиця 3

Час виконання типових логістичних запитів (мс)

СУБД	Q1: Оперативний (Last 10 items)	Q2: Аналітичний (Aggregated Mean)
Redis	**0.98**	**0.97**
PostgreSQL	1.67	0.88
TimescaleDB	2.24	1.64
InfluxDB	5.30	5.19

Джерело: власна розробка автора

Аналіз результатів читання:

1. **Ефективність Redis.** Для задач оперативного моніторингу (Q1) Redis є безальтернативним лідером із субмілісекундною затримкою (<1 мс). Це критично важливо для відображення руху транспорту на карті в реальному часі ("Real-time Tracking"), де затримка інтерфейсу понад 100 мс стає помітною для користувача.
2. **Порівняння SQL-рішень.** У тесті Q2 (агрегація середньої температури) PostgreSQL виявився швидшим (0.88 мс) за TimescaleDB (1.64 мс). Цей результат є специфічним для обсягів даних, що повністю вміщуються в оперативну пам'ять сервера. У цьому випадку накладні витрати TimescaleDB на сканування метаданих гіпертаблиці перевищують виграш від сканування лише окремих чанків. Проте, згідно з теоретичними моделями, при збільшенні обсягу даних до 100+ млн записів, продуктивність PostgreSQL різко знизиться через необхідність сканування всього індексу, тоді як TimescaleDB збереже стабільність завдяки механізму "chunk exclusion.
3. **Вплив протоколу InfluxDB.** Відносно висока затримка InfluxDB (~5 мс) пояснюється накладними витратами на мережевому рівні. На відміну від бінарних протоколів PostgreSQL та Redis, InfluxDB використовує HTTP API, що вимагає додаткового часу на встановлення TCP-з'єднання (handshake), передачу HTTP-заголовків та серіалізацію/десеріалізацію результату у формат JSON або CSV.

Порівняння ефективності використання дискового простору

Додатково було проведено оцінку ефективності зберігання даних на диску (Storage Efficiency) після запису 1.5 млн метрик.

- **InfluxDB** продемонстрував найвищий ступінь стиснення (приблизно 2-3 байт на точку) завдяки алгоритмам Gorilla Compression для чисел з плаваючою комою.
- **PostgreSQL та TimescaleDB** займають у 5-7 разів більше місця через фіксовану структуру рядка та накладні витрати на B-Tree індекси.
- **Redis**, працюючи в RAM, є найдорожчим рішенням з точки зору вартості зберігання одиниці інформації.

4. ОБГОВОРЕННЯ ТА ПРОПОЗИЦІЯ АРХІТЕКТУРИ

Результати показують, що жодна СУБД не є універсальною. InfluxDB ідеальний для запису ("write-heavy"), Redis – для читання ("read-heavy"), а TimescaleDB – для аналітики.

Для сучасних логістичних платформ, що інтегруються з ML та блокчейном, пропонується гібридну дворівневу архітектуру:

1. **Гарячий шар (Redis):** Використовується для відображення мапи в реальному часі. Зберігає дані лише за останні 24 години (TTL). Це забезпечує мінімальну затримку для диспетчерів.
2. **Холодний шар (TimescaleDB або InfluxDB):** Використовується для довготривалого зберігання.

o *TimescaleDB* рекомендовано для систем середнього навантаження (до 15k подій/сек), де важлива SQL-сумісність та складні JOIN-запити з бізнес-даними.

o *InfluxDB* рекомендовано для високонавантажених систем (>50k подій/сек), де пріоритетом є виключно збір метрик без складної реляційної логіки.

Висновки та перспективи подальших досліджень

У даній роботі вирішено актуальне науково-прикладне завдання обґрунтування архітектурного вибору стеку технологій для зберігання IoT-даних у логістичних системах. На основі проведеного порівняльного аналізу чотирьох класів СУБД (PostgreSQL, TimescaleDB, InfluxDB, Redis) можна зробити наступні висновки:

Ефективність архітектури для запису даних. Експериментально доведено, що для задач інтенсивного збору телеметрії (Write-Heavy workload) спеціалізовані NoSQL рішення на базі LSM-дерев (InfluxDB) демонструють перевагу над реляційними аналогами у 3–8 разів. Пікова пропускна здатність InfluxDB склала 121 800 повідомлень/сек, що робить її безальтернативним вибором для високонавантажених систем національного масштабу. Встановлено, що використання TimescaleDB доцільне лише при навантаженнях до 15 000 подій/сек; при перевищенні цього порогу накладні витрати на механізми партиціонування (chunk management) стають критичними, знижуючи продуктивність нижче рівня "чистого" PostgreSQL.

Обмеження In-Memory систем. Виявлено архітектурне обмеження СУБД Redis при сценарії масового запису різнорідних метрик. Незважаючи на роботу в оперативній пам'яті, однопоточна модель обробки подій (Single-threaded Event Loop) стає вузьким місцем при необхідності оновлення десятків тисяч ключів одночасно (15 833 повідомлень/сек). Таким чином, використання Redis як єдиного сховища для "сирих" історичних даних є неефективним як з точки зору швидкості запису, так і з точки зору вартості зберігання (RAM).

Оптимізація оперативного доступу. Для задач Real-time моніторингу (відображення геопозиції на карті, алерти диспетчера) Redis підтвердив статус найефективнішого рішення із латентністю менше 1 мс (0.98 мс). Жодна дискова СУБД (навіть з оптимізованим кешуванням) не здатна забезпечити аналогічний час відгуку.

Практичні рекомендації (Архітектурний патерн). На основі отриманих метрик запропоновано **гібридну Lambda-архітектуру** для логістичних платформ, яка розділяє потоки даних на два рівні:

Гарячий рівень (Speed Layer): Реалізується на базі **Redis TimeSeries**. Забезпечує миттєвий доступ до даних за останні 24 години. Це дозволяє нівелювати проблему повільного запису за рахунок меншого обсягу даних (TTL) та забезпечує інтерактивність UI.

Холодний рівень (Batch Layer): Реалізується на базі **TimescaleDB** (для систем середнього навантаження з потребою в SQL-аналітиці) або **InfluxDB** (для систем екстремального навантаження). Цей рівень відповідає за довгострокове зберігання, формування звітів та підготовку датасетів.

Перспективи досліджень. Подальший розвиток роботи вбачається у поєднанні запропонованої архітектури зберігання з методами штучного інтелекту та технологією блокчейн, що відповідає темі дисертаційного дослідження автора. Зокрема, планується дослідити:

1. Вплив стиснення даних у "холодному" сховищі на точність навчання нейромереж (Deep Learning) для задач прогнозування поломок транспорту.
2. Розробку механізму "якірної фіксації" (anchoring) хеш-сум часових рядів з TimescaleDB у приватний блокчейн (наприклад, Hyperledger Fabric) для створення незмінного аудиторського сліду перевезень без необхідності дублювання всього масиву даних у блокчейн.

Список використаної літератури

1. Gubbi J., Buyya R., Marusic S., Palaniswami M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems.* 2013. Vol. 29, No. 7. P. 1645–1660. URL: https://doi.org/10.1016/j.future.2013.01.010

2. Ramaswamy L., Lawson B., Gogineni S. V. *Towards a quality-centric big data architecture for federated sensor services.* Proceedings of the 2013 IEEE International Congress on Big Data (BigData Congress). IEEE, 2013. P. 86–93. URL: https://doi.org/10.1109/BigData.Congress.2013.21

3. Zaslavsky A., Perera C., Georgakopoulos D. Sensing as a service and big data. *arXiv preprint,* arXiv:1301.0159, 2013. URL: https://doi.org/10.48550/arXiv.1301.0159

4. Vongsingthong S., Smanchat S. A review of data management in Internet of Things. *KKU Research Journal*. 2015. Vol. 23. P. 215–240. URL: https://doi.nrct.go.th/ListDoi/listDetail?Resolve_DOI=10.14456/kkurj.2015.18

5. Ma M., Wang P., Chu C.-H. *Data management for Internet of Things: Challenges and opportunities.* Proceedings of the 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things. IEEE, 2013. P. 1144–1151. URL: https://doi.org/10.1109/GreenCom-iThings-CPSCom.2013.199

6. Zhu S. Creating a NoSQL database for the Internet of Things: Key-value store on the Sensei-Uring platform : Doctoral dissertation. Mid Sweden University, 2015. URL: https://www.diva-portal.org/smash/get/diva2:841604/FULLTEXT01.pdf

7. Bader A., Kopp O., Falkenthal M. Survey and comparison of open-source time series databases. *Datenbanksysteme für Business, Technologie und Web (BTW 2017) –* Workshopband. Bonn : Gesellschaft für Informatik, 2017. P. 249–268. URL: https://www.researchgate.net/publication/315838456_Survey_and_Comparison_of_Open_Source_Time_Series_Databases

8. Kiefer R. TimescaleDB vs. PostgreSQL for time-series: 20x higher inserts, 2000x faster deletes, 1.2x–14,000x faster queries. Timescale Blog, 2020. URL: https://blog.timescale.com

9. Fadhel M., Sekerinski E., Yao S. A comparison of time series databases for storing water quality data. *Auer M. E., Tsiatsos T. (eds.) Mobile Technologies and Applications for the Internet of Things (IMCL 2018).* Cham : Springer, 2019. P. 302–313. URL: https://doi.org/10.1007/978-3-030-11434-3_33

10. Grzesik P., Mrozek D. Comparative analysis of time series databases in the context of edge computing for low power sensor networks. *Computational Science – ICCS 2020. Lecture Notes in Computer Science,* Vol. 12141. Cham : Springer, 2020. P. 311–324. URL: https://doi.org/10.1007/978-3-030-50426-7_28

11. Dias L. B., Holanda M., Huacarpuma R. C., de Sousa Jr. R. T. *NoSQL database performance tuning for IoT data: Cassandra case study*. Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security (IoTBDS 2018). 2018. P. 277–284. URL: https://www.scitepress.org/papers/2018/67827/67827.pdf

References

1. Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems,* no. 29(7), pp. 1645–1660. URL: https://doi.org/10.1016/j.future.2013.01.010

2. Ramaswamy, L., Lawson, B., & Gogineni, S. V. (2013). *Towards a quality-centric big data architecture for federated sensor services*. In 2013 IEEE International Congress on Big Data (BigData Congress) (pp. 86–93). IEEE. URL: https://doi.org/10.1109/BigData.Congress.2013.21

3. Zaslavsky, A., Perera, C., & Georgakopoulos, D. (2013). Sensing as a service and big data. *arXiv preprint arXiv:1301.0159*. URL: https://doi.org/10.48550/arXiv.1301.0159

4. Vongsingthong, S., & Smanchat, S. (2015). A review of data management in Internet of Things. *KKU Research Journal*, no. 23, pp. 215–240. URL: https://doi.nrct.go.th/ListDoi/listDetail?Resolve_DOI=10.14456/kkurj.2015.18

5. Ma, M., Wang, P., & Chu, C.-H. (2013*). Data management for Internet of Things: Challenges and opportunities.* In Proceedings of the 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things (pp. 1144–1151). IEEE. URL: https://doi.org/10.1109/GreenCom-iThings-CPSCom.2013.199

6. Zhu, S. (2015). *Creating a NoSQL database for the Internet of Things: Key-value store on the Sensei-Uring platform* (Doctoral dissertation, Mid Sweden University). URL: https://www.diva-portal.org/smash/get/diva2:841604/FULLTEXT01.pdf

7. Bader, A., Kopp, O., & Falkenthal, M. (2017). Survey and comparison of open-source time series databases. In *Datenbanksysteme für Business, Technologie und Web (BTW 2017) – Workshopband* (pp. 249–268). Bonn: Gesellschaft für Informatik. URL: https://www.researchgate.net/publication/315838456_Survey_and_Comparison_of_Open_Source_Time_Series_Databases

8. Kiefer, R. (2020). *TimescaleDB vs. PostgreSQL for time-series: 20x higher inserts, 2000x faster deletes, 1.2x–14,000x faster queries*. Timescale Blog. Retrieved from https://blog.timescale.com

9. Fadhel, M., Sekerinski, E., & Yao, S. (2019). A comparison of time series databases for storing water quality data. In M. E. Auer & T. Tsiatsos (Eds.), *Mobile Technologies and Applications for the Internet of Things (IMCL 2018)* (pp. 302–313). Cham: Springer. URL: https://doi.org/10.1007/978-3-030-11434-3_33

10. Grzesik, P., & Mrozek, D. (2020). Comparative analysis of time series databases in the context of edge computing for low power sensor networks. In *Computational Science – ICCS 2020* (Lecture Notes in Computer Science, no. 12141, pp. 311–324). Cham: Springer. URL: https://doi.org/10.1007/978-3-030-50426-7_28

11. Dias, L. B., Holanda, M., Huacarpuma, R. C., & de Sousa Jr, R. T. (2018). NoSQL database performance tuning for IoT data: Cassandra case study. In *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security (IoTBDS 2018)* (pp. 277–284). URL: https://www.scitepress.org/papers/2018/67827/67827.pdf

Дата першого надходження рукопису до видання: 21.11.2025
Дата прийнятого до друку рукопису після рецензування: 17.12.2025
Дата публікації: 31.12.2025