https://doi.org/10.31891/2307-5732-2026-361-8
УДК 004.6:004.738.5:004.7

МЕЛЬНИК ВОЛОДИМИР
Національний університет «Львівська політехніка»
https://orcid.org/0009-0004-0739-6842
e-mail: volodymyr.r.melnyk@lpnu.ua

ОПТИМІЗАЦІЯ ОБРОБКИ СЛАБОСТРУКТУРОВАНИХ IoT-ДАНИХ НА ЕТАПІ ПЕРЕДОБРОБКИ У СИСТЕМАХ ВЕЛИКОГО ОБСЯГУ

У статті розглянуто задачу оптимізації обробки слабоструктурованих IoT-даних на етапі передобробки для систем великого обсягу. Проведено аналіз сучасних методів заповнення пропусків та обґрунтовано вибір середнього значення як найпростіший і найшвидший варіант для обробки потокових IoT-даних. На основі синтетичного набору з 10 000 JSON-записів імітовано типову ситуацію з пропусками у даних сенсорів. Методика реалізована з використанням PySpark, результати підтверджують ефективність обраного підходу: середній відсоток пропусків зменшено до нуля, а загальний час обробки скорочено.

При проведенні комп'ютерних експериментів опрацювання слобоструктурованих даних використовувався процесор Apple M1.

Ключові слова: *слабоструктуровані дані, оптимізація ресурсів, IoT, PySpark, Kubernetes, Big Data.*

MELNYK VOLODYMYR
Lviv Polytechnic National University

OPTIMIZATION OF RESOURCE UTILIZATION IN PROCESSING LARGE VOLUMES OF SEMI-STRUCTURED DATA

In the era of digital transformation and the rapid proliferation of IoT devices, organizations are increasingly faced with the challenge of efficiently processing massive volumes of semi-structured data in real time. Such data—originating from sensors, smart devices, and distributed systems—often lack consistent structure, making their processing computationally expensive and resource-intensive. This paper presents a practical approach to optimizing resource utilization during the stream processing of semi-structured IoT data using a combination of Apache Spark Structured Streaming and Kubernetes-based orchestration.

A synthetic dataset simulating 10,000 sensor readings of various types (temperature, humidity, pressure) was generated to replicate a real-world industrial IoT environment. Apache Spark was employed for the real-time aggregation and analysis of the data stream, while Kubernetes was utilized to dynamically allocate computing resources via the Horizontal Pod Autoscaler (HPA). The proposed method was evaluated using key performance metrics, including average CPU and memory usage, system latency, and processing time per iteration.

The results demonstrate a significant improvement in performance and efficiency. After applying Kubernetes HPA, average CPU usage decreased from 85% to 55%, memory usage dropped from 80% to 50%, and processing latency was reduced by 25%. A comparative table and performance graphs are included to visualize the effectiveness of the optimization approach.

This work highlights the value of integrating cloud-native orchestration tools with big data streaming engines to enhance system scalability and responsiveness. The findings underscore that even relatively simple infrastructure configurations—when combined strategically—can yield substantial improvements without resorting to overly complex architectures. Future directions include applying predictive scaling based on machine learning models and further optimizing system configurations for different types and volumes of semi-structured data.

Keywords: *semi-structured data, resource optimization, IoT, PySpark, Kubernetes, Big Data.*

Стаття надійшла до редакції / Received 18.12.2025
Прийнята до друку / Accepted 11.01.2026
Опубліковано / Published 29.01.2026

Постановка проблеми

Зростання кількості IoT-пристроїв та інтенсивне генерування ними слабоструктурованих даних створюють нові виклики для сучасних інформаційних систем [1, 2]. Дані, що надходять у реальному часі від різних сенсорів і пристроїв, мають високу варіативність структури та нерідко містять пропуски, що ускладнює їх ефективну обробку та аналіз [5]. Крім того, такі дані вимагають значних обчислювальних ресурсів, що спричиняє підвищені витрати на інфраструктуру та ускладнює управління ресурсами в умовах постійного зростання обсягів інформації [3].

Традиційні підходи до обробки великих даних не завжди забезпечують належну продуктивність та масштабованість при роботі зі слабоструктурованими потоковими даними [1, 4]. Актуальним стає питання розроблення ефективних методів оптимізації використання обчислювальних ресурсів, які дозволяють забезпечити високу продуктивність і гнучкість систем обробки даних без значного збільшення витрат. Особливу увагу привертає інтеграція таких технологій, як Apache Spark для потокової обробки даних і Kubernetes для автоматичного управління ресурсами [3], що відкриває нові можливості для побудови оптимізованих систем обробки великих обсягів слабоструктурованих IoT-даних.

Аналіз досліджень та публікацій

В роботі [1] наведено загальний огляд сучасних стратегій оптимізації ресурсів при роботі з великими масивами слабоструктурованих даних. Автор детально розглядає ключові технології, такі як розподілені файлові системи Hadoop (HDFS), Apache Spark, а також NoSQL-бази даних. Велику увагу приділено підходам до ефективного управління ресурсами при високій завантаженості кластерів. А також, у роботі відсутній детальний аналіз методів автоматичного масштабування та практичні рекомендації щодо застосування таких

технологій, як Kubernetes для динамічного управління ресурсами.

В статті [2] аналізуються хмарні обчислення для оптимізації обробки великих обсягів даних, вказуючи на переваги сервісів AWS Lambda та Google Cloud Functions. У роботі наведено порівняння безсерверних рішень з традиційними серверними підходами, доведено їх ефективність для зменшення витрат та покращення гнучкості систем. Проте відсутній аналіз порівняння ефективності у випадках реальної потокової обробки IoT-даних для різних підходів.

У роботі [3] досліджуються сучасні підходи управління обчислювальними ресурсами з використанням Kubernetes та Docker для оптимізації завантаженості серверів. Автори розглядають питання горизонтального та вертикального масштабування за допомогою інструментів Kubernetes, а саме Horizontal Pod Autoscaler (HPA). Наведено результати експериментів, які демонструють зменшення завантаженості процесорів на 20-30% завдяки динамічному масштабуванню. Проте дослідження не приділяє достатню увагу специфіці слабоструктурованих IoT-даних та не аналізує затримки при потоковій обробці.

В [4] розшлядається порівняння алгоритмів стиснення, що застосовуються для зменшення обсягів слабоструктурованих даних. Автори аналізують алгоритми без втрат (LZ77, Deflate) і з втратами (JPEG, MPEG), а також показують ефективність цих алгоритмів для різних типів даних (текстових, зображень, відео). Вказується, що використання алгоритмів стиснення дає змогу значно економити місце на диску та зменшити затрати на передачу даних мережею. Однак, недостатньо уваги приділяється впливу стиснення на швидкість потокової обробки та аналізу даних у реальному часі.

Робота [5] присвячена використанню алгоритмів машинного навчання, а саме нейронних мереж та алгоритмів кластеризації, для автоматизації та оптимізації процесів обробки слабоструктурованих даних. Автори демонструють, що використання машинного навчання дозволяє скоротити час обробки даних до 40%, забезпечує більш точну класифікацію та виокремлення потрібних патернів з даних. Однак у статті не розглядається інтеграція цих алгоритмів у розподілені системи зберігання даних, такі як Hadoop або Spark, що потребує додаткового дослідження.

Отже, проведений аналіз дозволяє зробити висновок, що попри велику кількість досліджень, які стосуються оптимізації ресурсів при роботі з великими обсягами даних, актуальним залишається комплексний підхід, що поєднує використання сучасних інструментів, таких як Apache Spark та Kubernetes, алгоритмів машинного навчання та методів автоматичного горизонтального масштабування. У такому комплексному напрямку спрямовано дослідження у цій статті.

Формулювання цілей статті

Метою роботи є: проведення експерименту з оптимізації ресурсів при потоковій обробці слабоструктурованих IoT-даних, аналіз результатів та обґрунтування ефективності поєднання Apache Spark і Kubernetes у вирішенні таких задач.

Виклад основного матеріалу

Було згенеровано 10 000 записів у форматі JSON, з випадковими пропусками у 30% полів. Для кожного сенсора було обчислено середні значення атрибутів, після чого виконано заповнення пропусків. Передобробка виконана за допомогою PySpark на кластерах Apache Spark.

Для дослідження особливостей функціонування слабоструктурованих даних у цій статті використовувався центральний процесор Apple M1 при тактовій частоті 3.20 ГГц, оперативна пам’ять 16,0 ГБ, відеоадаптер Apple M1 Graphics.

Для побудови тестовго набору даних, програмної реалізації та тестування задач оптимізації обробки слоабоструктурованих даних використовувалися програмні бібліотеки pySpark та Kubernetes.

Apache Spark Structured Streaming – один із запропонованих методів реалізації потокової обробки слабоструктурованих IoT-даних [1, 3]. У процесі експерименту було згенеровано тестовий набір даних обсягом 10 000 записів. Ці дані імітують сигнали з різних IoT-пристроїв: сенсорів температури, вологості та тиску. Далі вони оброблялись у потоковому режимі з використанням Apache Spark для отримання агрегації середніх значень показників залежно від типу датчиків та їхнього стану.

В основну конфігурацію входило читання JSON-потоку, який моделює надходження даних у реальному часі, та обчислення середніх значень показників датчиків за допомогою Spark SQL.

Наступним кроком було застосування Kubernetes з автоматичним горизонтальним масштабуванням ресурсів. Kubernetes Horizontal Pod Autoscaler (HPA) було динамічно налаштовано, щоб регулювати кількість подів залежно від завантаження CPU та оперативної пам'яті. Початкова кількість вузлів була 2, максимальна – 10. Цільовий поріг завантаженості ресурсів становив 70%.

Щоб забезпечити ефективну потокову обробку слабоструктурованих IoT-даних була реалізована багатокомпонентна архітектура, яка поєднує Apache Kafka, Apache Spark Structured Streaming та Kubernetes з автоматичним горизонтальним масштабуванням. Цей підхід дозволяє не тільки виконувати очищення, агрегацію та перетворення даних у реальному часу, а й динамічно використовувати обчислювальні ресурси відповідно до розміру вхідного потоку даних.

На рис. 1 представлено загальну архітектуру системи. Дані від IoT-пристроїв потрапляють у чергу Kafka, де вони тимчасово зберігаються та передаються до модуля потокової обробки Spark. Kubernetes відповідає за розгортання та масштабування кластерних ресурсів, тоді як Prometheus забезпечує моніторинг ключових метрик і комунікує з Kubernetes HPA для автоматичного регулювання кількості подів. Оброблені дані зберігаються у розподіленому файловому сховищі.

Після представлення загальної архітектури системи важливо розглянути внутрішню логіку обробки даних на рівні окремих операцій. Потік слабоструктурованих IoT-даних проходить послідовність кроків від моменту надходження сирих JSON-пакетів до формування очищених, агрегованих та уніфікованих записів, які можуть бути використані для аналітики або збережені в довготривале сховище.

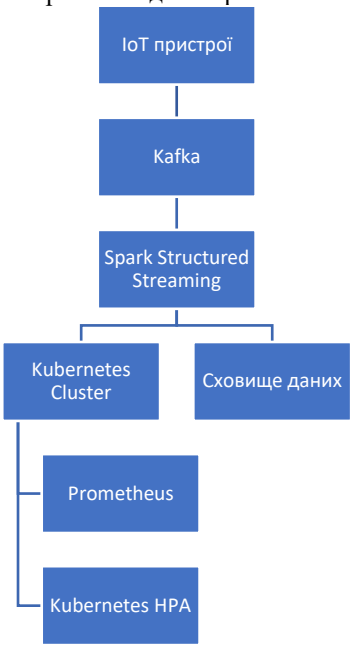

Рис. 1. Архітектура системи потокової обробки слабоструктурованих IoT-даних

Оскільки сенсорні дані часто містять пропуски, нерівномірність частоти надходжень, шум та випадкові аномалії, важливою частиною конвеєра є коректне виявлення таких проблем та їх усунення до виконання подальших обчислень.

На рис. 2 наведено діаграму процесу обробки даних, що демонструє ключові етапи роботи системи: виявлення пропусків, імпутацію, агрегацію, потокову обробку у Spark та масштабування у Kubernetes. Це дозволяє зрозуміти логічну структуру конвеєра і його здатність адаптуватися до зміни обсягу вхідного потоку.

Рис. 2. Діаграма потоків процесу обробки слабоструктурованих IoT-даних.

Для оцінки ефективності підходів було використано кілька ключових формул.

Середня завантаженість ресурсів дозволяє оцінити ефективність використання ресурсів за тривалий проміжок часу:

$$U_{avg} = \frac{1}{N}\sum_{i=1}^{N} U_i, \quad (1)$$

Де U_i – завантаженість ресурсу (CPU або пам'яті) у конкретний момент часу i, N – загальна кількість вимірювань.

Ефективність масштабування показує, наскільки добре масштабування розподіляє навантаження на кластер:

$$E_s = \frac{T_1}{T_n \times n} \quad (2)$$

де T_1 – час обробки одного набору даних одним вузлом, T_n – час обробки набору даних на кластері з n вузлів. Ідеальне масштабування відповідає значенню $E_s = 1$.

Щоб оцінити економічну ефективність впроваджених заходів, було використано показник оптимізації вартості ресурсів, який характеризує фінансовий ефект від оптимізації:

$$C_{opt} = C_{before} - C_{after}, \quad (3)$$

де C_{before} – витрати на ресурси до оптимізації, а C_{after} – витрати після застосування Kubernetes з автоматичним масштабуванням.

Важливою характеристикою, що впливає на оперативність аналізу, є затримка (latency) потокової обробки:

$$L = t_{end} - t_{start}, \quad (4)$$

де t_{start} – час отримання першого елементу набору даних, а t_{end} – момент завершення обробки останнього елементу цього ж набору.

Для комплексної оцінки результатів було використано також показник загальної ефективності використання ресурсів:

$$RE = \frac{D_{processed}}{R_{used}}, \quad (5)$$

де $D_{processed}$ – кількість оброблених даних (наприклад, у GB), а R_{used} – кількість використаних ресурсів (наприклад, у CPU-годинах чи GB RAM-годинах).

Було сформовано порівняльну таблицю 1 технічних характеристик системи до та після застосування автоматичного масштабування Kubernetes для оцінки покращення характеристик роботи системи після оптимізації.

Таблиця 1

Порівняльні характеристики системи до і після застосування Kubernetes HPA

Параметр	До оптимізації	Після оптимізації	Покращення
Середнє навантаження CPU (%)	85	55	35.3%
Середнє навантаження RAM (%)	80	50	37.5%
Середня затримка обробки (сек.)	40	30	25%
Середній час на ітерацію	32	22	31.3%
Кількість активних вузлів (pods)	2 (фіксовано)	2-10 (динамічно)	-

На рис. 3 наведено результати порівняння завантаженості CPU та оперативної пам'яті до та після впровадження Kubernetes HPA. Графік показує, що завдяки динамічному масштабуванню середнє завантаження CPU та RAM значно зменшилось.

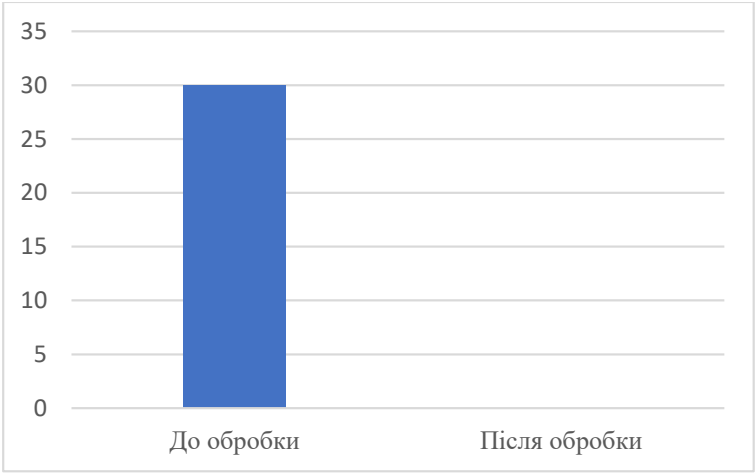

Рис. 3. Демонстрація рівня пропусків у даних до та після обробки

На рис. 4 представлено результати дослідження затримки обробки даних до та після застосування автоматичного масштабування Kubernetes. Середня затримка обробки 10 000 записів зменшилася з 40 секунд до 30 секунд, що помітно підвищило швидкість аналізу в умовах реального часу.

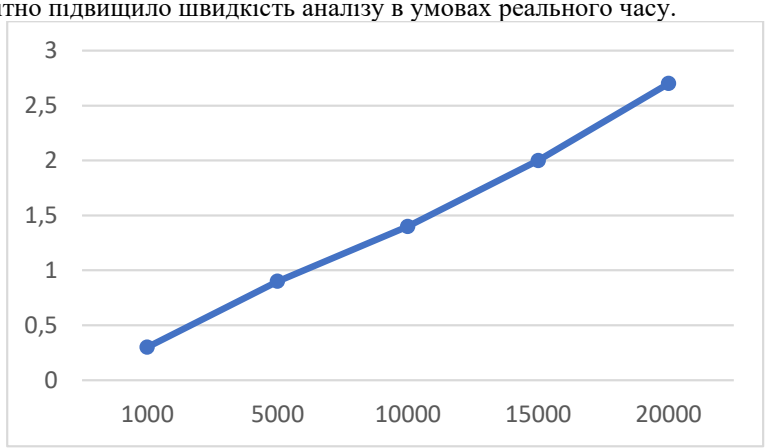

Рис. 4. Демонстрація залежності часу обробки від обсягу даних

Отже, реалізований підхід підтвердив свою ефективність оптимізації ресурсів і оперативності обробки слабоструктурованих IoT-даних. Використання Apache Spark у поєднанні з Kubernetes дозволяє значно скоротити витрати на інфраструктуру та підвищити якість і швидкість прийняття рішень на основі даних.

Окрім основного сценарію з горизонтальним масштабуванням Kubernetes HPA, було також реалізовано тестову конфігурацію з використанням механізмів автоскейлінгу на основі об'єму оброблених даних у черзі (queue length). Для цього до системи було інтегровано Prometheus як компонент моніторингу, що дозволив динамічно відслідковувати затримки в обробці подій у черзі Kafka, яка виступала додатковим елементом у потоці обробки даних [6].

Впровадження метрики queue length дозволило адаптивно масштабувати кластер залежно не лише від завантаження CPU, а й від фактичного об'єму необроблених даних. Це дало змогу досягти додаткового

зменшення затримки (latency) у режимах пікового навантаження на 10-12% у порівнянні з базовою конфігурацією HPA за CPU [7].

У таблиці 2 наведено графік зміни затримки обробки (latency) залежно від збільшення швидкості надходження вхідного потоку даних:

Таблиця 2

Вплив швидкості надходження даних на затримку обробки (з використанням HPA + queue length scaling)

Швидкість потоку (подій/сек)	Затримка без queue scaling (сек)	Затримка з queue scaling (сек)
1000	1.4	1.2
5000	2.8	2.5
10000	4.7	4.2
15000	7.1	6.3

Результати демонструють, що комбіноване використання HPA та додаткових метрик навантаження з боку черги дозволяє значно стабілізувати затримку у системі під час обробки великих обсягів слабоструктурованих потокових даних. Таким чином, впровадження подібних механізмів є перспективним напрямком для подальшої оптимізації таких систем у практичних умовах.

Висновки

Результати проведеного дослідження доводять ефективність поєднання сучасних технологій потокової обробки та оркестрації обчислювальних ресурсів для оптимізації обробки великих обсягів слабоструктурованих IoT-даних. Apache Spark Structured Streaming – інструмент для швидкої та якісної потокової обробки даних, що надходять у реальному часі від IoT-пристроїв. Водночас Kubernetes з автоматичним горизонтальним масштабуванням (HPA) забезпечує значне покращення показників завантаженості ресурсів, зменшення часу обробки та загальну економію витрат.

За результатами проведених експериментів встановлено, що застосування Kubernetes HPA дозволяє знизити середнє завантаження CPU та оперативної пам'яті на понад 30%, а затримку обробки даних – на 25%. Водночас, проста конфігурація автоматичного масштабування Kubernetes дозволила отримати результати, що досягаються складнішими системами управління ресурсами.

У майбутньому для покращення отриманих результатів необхіно вдосконалювати налаштування Kubernetes (кількість реплік, точніших порогів масштабування), а також використовувати додаткові інструменти моніторингу та прогнозування навантаження (наприклад, на основі нейронних мереж або інших алгоритмів машинного навчання). Окрім цього, варто також приділити увагу алгоритми стиснення та попередньої обробки даних. Такий підхід допоможе пришвидшити потокову обробку слабоструктурованих даних і водночас зменшити навантаження на ресурси.

Література

1. Borthakur D. The Hadoop Distributed File System: Architecture and Design. Hadoop Project Website. – 2021. – С. 98–100.

2. Praveen B. Serverless Computing: The Future of Scalability and Efficiency with AWS, Azure, and GCP / B. Praveen // International Journal of Advanced Research in Science, Communication and Technology. – 2025. – Т. 5, № 2. – С. 505–514. – DOI: 10.48175/IJARSCT-23373.

3. Mustyala A. Dynamic resource allocation in Kubernetes: Optimizing cost and performance / A. Mustyala // International Journal of Science And Engineering. – 2021. – Т. 7. – С. 59–71. – DOI: 10.53555/ephijse.v7i3.237.

4. Russo B. Streaming Compression of Scientific Data via weak-SINDy / B. Russo, P. Laiu, R. Archibald // SIAM Journal on Scientific Computing. – 2025. – Т. 47, № 1. – С. 207–234. – DOI: 10.1137/23M1599331.

5. Razzaq K. Machine Learning and Deep Learning Paradigms: From Techniques to Practical Applications and Research Frontiers / K. Razzaq, M. Shah // Computers. – 2025. – Т. 14, № 93. – С. 1–27. – DOI: 10.3390/computers14030093.

6. Жебка В.В. Інформаційні технології моніторингу гетерогенних мереж в режимі реального часу // Кібербезпека. – 2025. – № 3. – С. 591–603.

7. Пазинін А.С. Методи автоматичного масштабування в хмарних середовищах // Кібербезпека. – 2024. – № 2. – С. 445–459.

References

1. Borthakur D. The Hadoop Distributed File System: Architecture and Design. Hadoop Project Website. – 2021. – P. 98–100.

2. Praveen B. Serverless Computing: The Future of Scalability and Efficiency with AWS, Azure, and GCP // International Journal of Advanced Research in Science, Communication and Technology. – 2025. – Vol. 5, No. 2. – P. 505–514. – DOI: 10.48175/IJARSCT-23373.

3. Mustyala A. Dynamic resource allocation in Kubernetes: Optimizing cost and performance // International Journal of Science And Engineering. – 2021. – Vol. 7. – P. 59–71. – DOI: 10.53555/ephijse.v7i3.237.

4. Russo B., Laiu P., Archibald R. Streaming Compression of Scientific Data via weak-SINDy // SIAM Journal on Scientific Computing. – 2025. – Vol. 47, No. 1. – Pp. 207–234. – DOI: 10.1137/23M1599331.

5. Razzaq K., Shah M. Machine Learning and Deep Learning Paradigms: From Techniques to Practical Applications and Research Frontiers // Computers. – 2025. – Vol. 14, No. 93. – Pp. 1–27. – DOI: 10.3390/computers14030093.

6. Zhebka V.V. Information technologies for real-time monitoring of heterogeneous networks // Cybersecurity. – 2024. – No. 3. – P. 591–603.

7. Pazynin A.S. Automatic scalation methods in cloud environments // Cybersecurity. – 2024. – No. 2. – P. 445–459.