https://doi.org/10.31891/2307-5732-2026-365-56
УДК 004.75:004.77:004.415

ГАДЬО ІРИНА
Національний університет "Львівська політехніка"
https://orcid.org/0000-0003-1615-6483
e-mail: iryna.v.nychai@lpnu.ua
ШУЛЯК НАЗАР
Національний університет "Львівська політехніка"
e-mail: nazar.shuliak.oi.2022@lpnu.ua
ЛЯХ ІГОР
Ужгородський національний університет
https://orcid.org/0000-0001-5417-9403
e-mail: igor.lyah@uzhnu.edu.ua

ЗМЕНШЕННЯ РЕБАЛАНСУВАННЯ В ПОТОКОВИХ АРХІТЕКТУРАХ НА БАЗІ APACHE KAFKA

У роботі досліджено проблему забезпечення стабільності функціонування потокових систем у динамічних середовищах. Розглянуто особливості архітектури систем на базі Apache Kafka та механізми координації груп споживачів, зокрема процес ребалансування партицій. Показано, що часті зміни складу споживачів, характерні для мікросервісних і хмарних середовищ, призводять до частих ініціацій ребалансування, що супроводжується перериваннями обробки даних і зростанням затримок.

Запропоновано підхід до зменшення кількості надлишкових ребалансувань, що базується на координації життєвого циклу споживачів. На відміну від існуючих рішень, орієнтованих на оптимізацію алгоритмів розподілу партицій, запропонований підхід спрямований на зменшення частоти виникнення ребалансування шляхом узгодження процесів підключення та відключення споживачів.

Проведений аналіз показав, що застосування запропонованого підходу дає змогу підвищити стабільність розподілу партицій, зменшити кількість переривань у обробці даних та покращити ефективність функціонування потокових систем у реальному часі.

Ключові слова: *потокові системи; Apache Kafka; ребалансування; групи споживачів; розподілені системи; обробка даних у реальному часі*

IRYNA GADO, NAZAR SHULIAK
National University "Lviv Polytechnic"
LIAKH IHOR
Uzhhorod National University

REDUCING REBALANCING IN STREAMING ARCHITECTURES BASED ON APACHE KAFKA

This paper addresses the problem of ensuring the stability of streaming systems operating in dynamic and highly scalable environments. The architectural characteristics of systems based on Apache Kafka are analyzed, with particular attention given to consumer group coordination mechanisms and the partition rebalancing process. It is demonstrated that frequent changes in the composition of consumer groups, which are typical for cloud-native and microservice-based architectures, lead to repeated rebalancing events. These events cause temporary suspension of data processing, increased latency, and reduced throughput, ultimately affecting the overall system performance. The study examines the underlying causes of excessive rebalancing, including autoscaling, service restarts, and deployment updates, which trigger frequent join and leave operations within consumer groups. Existing approaches primarily focus on optimizing partition assignment strategies, such as range, round-robin, and cooperative rebalancing, aiming to reduce data movement during rebalancing. However, they do not sufficiently address the root cause of frequent rebalancing initiation. To overcome these limitations, an approach based on coordinated lifecycle management of consumers is proposed. The core idea of the approach is to reduce the number of events that trigger rebalancing by synchronizing consumer join and leave operations. This includes controlled startup of new consumers, delayed shutdown to allow completion of in-progress processing, and explicit coordination of group membership changes. The proposed method does not require modification of the underlying Kafka mechanisms and can be integrated into existing distributed systems. The analysis shows that the implementation of the proposed approach leads to improved stability of partition distribution, reduced frequency of rebalancing events, and minimized processing interruptions. As a result, the efficiency and reliability of real-time streaming systems are significantly enhanced. The findings highlight the importance of managing consumer lifecycle events as a key factor in achieving stable and efficient stream processing in dynamic environments.

Keywords: *streaming systems; Apache Kafka; rebalancing; consumer groups; distributed systems; real-time data processing*

Стаття надійшла до редакції / Received 11.02.2026
Прийнята до друку / Accepted 11.03.2026
Опубліковано / Published 28.05.2026

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

Сучасні інформаційні системи дедалі частіше функціонують в умовах необхідності обробки даних у режимі реального часу, що характерно для задач аналізу потокових даних, обробки текстових повідомлень, моніторингу подій та логування у розподілених середовищах. Для реалізації таких систем широко застосовуються платформи потокової обробки даних, зокрема Apache Kafka, яка забезпечує масштабованість, відмовостійкість та високу пропускну здатність.

Одним із ключових механізмів організації обробки даних у таких системах є використання груп

споживачів, що дає змогу здійснювати паралельну обробку повідомлень. Стабільність функціонування потокових систем значною мірою залежить від ефективного та узгодженого розподілу партицій між споживачами.

У процесі експлуатації потокових систем відбуваються часті зміни складу споживачів, зумовлені масштабуванням сервісів, оновленням програмного забезпечення або відмовами окремих компонентів. У таких умовах ініціюється процедура ребалансування групи споживачів, що передбачає перерозподіл партицій між активними екземплярами.

Незважаючи на функціональну необхідність ребалансування, цей процес супроводжується тимчасовим призупиненням обробки повідомлень, що призводить до збільшення затримок та зниження пропускної здатності системи. У динамічних середовищах, де зміни складу споживачів відбуваються регулярно, часті ребалансування можуть спричиняти нестабільність потокових процесів і порушення безперервності обробки даних.

Таким чином, актуальною науково-практичною задачею є зменшення кількості надлишкових ребалансувань у групах споживачів та підвищення стабільності функціонування потокових систем на базі Apache Kafka в умовах динамічних середовищ. Розв'язання цієї задачі є важливим для забезпечення ефективної та безперервної обробки даних у системах реального часу.

Аналіз досліджень та публікацій

Розподілені платформи потокової обробки подій є предметом активних досліджень у контексті забезпечення масштабованої та низьколатентної обробки даних. Архітектури на базі Apache Kafka широко застосовуються у системах реального часу, що підтверджується сучасними дослідженнями у сфері потокової аналітики [1–3].

У наукових роботах значну увагу приділено питанням масштабованості потокових систем, зокрема механізмам партиціювання та балансування навантаження. Дослідження [3–5] демонструють, що ефективний розподіл партицій є критичним для забезпечення високої продуктивності, тоді як нераціональна конфігурація може призводити до нерівномірного використання ресурсів і зростання затримок.

Окремий напрям досліджень стосується алгоритмів призначення партицій у межах груп споживачів. У роботах [2, 3, 6] проаналізовано різні стратегії розподілу (range, round-robin, sticky) та їх вплив на стабільність обробки даних. Показано, що хоча ці підходи спрямовані на покращення балансування, процес ребалансування все ще супроводжується тимчасовими перервами в обробці повідомлень.

Для зменшення негативного впливу ребалансування досліджуються підходи, орієнтовані на мінімізацію переміщення партицій та підвищення стабільності їх розподілу. У роботах [5–7] запропоновано адаптивні алгоритми та механізми планування, що дають змогу зменшити простої споживачів і підвищити стабільність пропускної здатності системи.

Також активно досліджується проблема еластичного масштабування потокових систем. У роботах [8–11] запропоновано методи автоматичного регулювання кількості споживачів відповідно до змін навантаження, що дає змогу ефективно використовувати ресурси системи. Водночас такі підходи можуть призводити до частих змін складу груп споживачів.

У контексті сучасних мікросервісних і хмарних архітектур додатковою проблемою є часті перезапуски та заміна сервісів, що зумовлює регулярні зміни складу споживачів. Існуючі механізми, зокрема статичне членство та кооперативне ребалансування, сприяють частковому зменшенню впливу цих змін, однак не усувають проблему повністю.

Таким чином, існуючі дослідження переважно зосереджені на вдосконаленні алгоритмів розподілу партицій та механізмів масштабування, тоді як питання координації життєвого циклу споживачів у динамічних середовищах отримало менше уваги. Зокрема, обмежена увага приділяється підходам, що дозволяють узгоджувати процеси підключення та відключення споживачів з метою мінімізації надлишкових ребалансувань.

Формулювання цілей статті

Метою даної роботи є підвищення стабільності функціонування потокових систем на базі Apache Kafka шляхом зменшення кількості надлишкових ребалансувань груп споживачів на основі координації їх життєвого циклу в динамічних середовищах.

Для досягнення поставленої мети у роботі вирішуються такі завдання:

- аналіз архітектури потокових систем на основі Apache Kafka;
- дослідження механізмів розподілу партицій і координації груп споживачів;
- аналіз причин виникнення ребалансування у динамічних середовищах;
- дослідження впливу частих ребалансувань на безперервність обробки даних;
- розробка підходу до координації життєвого циклу споживачів з метою зменшення кількості надлишкових ребалансувань.

Виклад основного матеріалу

Потокові системи на базі Apache Kafka реалізують обробку даних у режимі реального часу за допомогою розподіленої архітектури, що включає джерела даних (producer), брокери повідомлень та споживачів (consumer). Повідомлення публікуються у топіки, які поділяються на партиції, що забезпечує можливість

паралельної обробки даних. Споживачі об'єднуються у групи, в межах яких кожна партиція призначається лише одному споживачу. Такий підхід дає змогу досягти балансування навантаження та масштабування обробки шляхом зміни кількості екземплярів споживачів.

У динамічних середовищах, зокрема в умовах мікросервісної архітектури, склад груп споживачів може змінюватися внаслідок масштабування сервісів, їх перезапуску або оновлення. Це зумовлює необхідність адаптивного перерозподілу партицій між споживачами. У системах на базі Apache Kafka зміна складу групи споживачів ініціює процес ребалансування (рис. 1), під час якого виконується повторний розподіл партицій між активними споживачами. Така процедура є необхідною для забезпечення узгодженості обробки та ефективного використання ресурсів. Процес ребалансування включає етапи зупинки обробки повідомлень, відкликання партицій у споживачів та їх повторного призначення. У цей період обробка даних тимчасово призупиняється, що призводить до накопичення повідомлень у черзі та збільшення затримок.

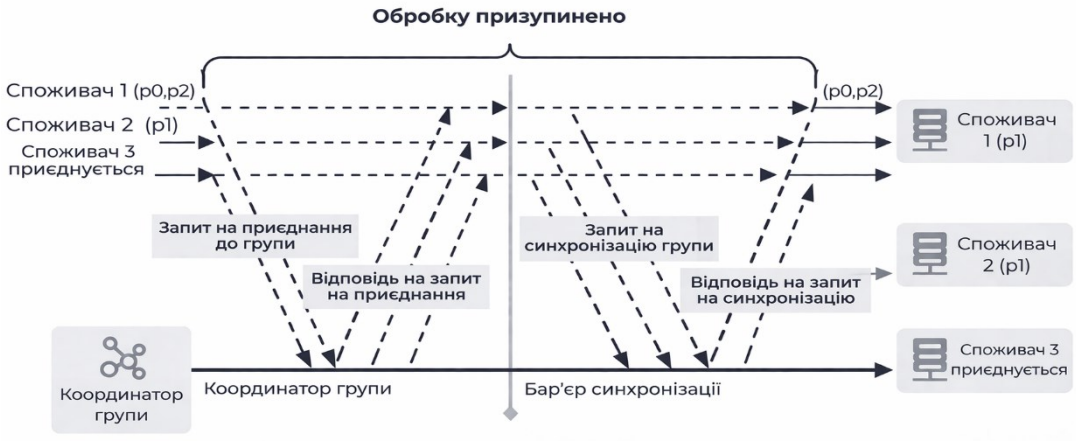

Рис. 1. Процес ребалансування [12]

Аналіз практичного функціонування систем показує, що у динамічних середовищах ребалансування може ініціюватися не лише через відмови, але й через керовані зміни, зокрема масштабування або оновлення сервісів. Часті ініціації цього процесу призводять до нестабільності обробки та зниження загальної продуктивності системи.

Таким чином, навіть за наявності оптимізованих алгоритмів розподілу партицій, сам факт частого виникнення ребалансування є критичним фактором, що впливає на ефективність потокових систем.

З метою зменшення кількості надлишкових ребалансувань у роботі запропоновано підхід, що базується на координації життєвого циклу споживачів у групі.

Основна ідея підходу полягає у зменшенні кількості подій, що ініціюють ребалансування, шляхом контрольованого підключення та відключення споживачів. Зокрема, пропонується уникати одночасних змін складу групи та забезпечувати послідовну взаємодію споживачів із системою.

Підхід передбачає:

- контрольований запуск нових споживачів із попередньою синхронізацією стану;
- відкладене відключення споживачів до завершення обробки призначених їм партицій;
- узгодження змін складу групи з поточним станом обробки даних.

Для формалізації запропонованого підходу розглянемо етапи координації життєвого циклу споживачів, а також їх вплив на характеристики потокової системи (табл. 1).

Таблиця 1

Етапи координації життєвого циклу споживачів та їх вплив на характеристики системи

Етап	Опис дії	Без підходу	З підходом
1	Виявлення зміни складу групи споживачів	Негайна реакція системи	Контрольована обробка події
2	Обробка запиту на приєднання/вихід	Миттєвий join/leave	Відкладення операції
3	Стан обробки повідомлень	Призупинення обробки	Завершення поточних задач
4	Синхронізація групи	Відсутня або часткова	Явна синхронізація перед зміною
5	Ініціація ребалансування	Часта	Зменшена кількість
6	Розподіл партицій	Повний перерозподіл	Мінімізована міграція
7	Відновлення обробки	Із затримками	Швидке та стабільне
8	Загальна стабільність системи	Низька у динамічних умовах	Підвищена

Як видно з табл. 1, запропонований підхід дає змогу зменшити частоту ініціації ребалансування та підвищити стабільність функціонування системи без модифікації базових механізмів Apache Kafka. Отримані

результати мають аналітичний характер та базуються на узагальненні поведінки системи, спостереженої у практичних сценаріях.

Сучасні механізми, реалізовані в Apache Kafka, зокрема статичне членство споживачів та кооперативне ребалансування, спрямовані на зменшення впливу процесу ребалансування на обробку даних. Однак ці підходи не усувають сам факт ініціації ребалансування при зміні складу групи. На відміну від них, зосереджених переважно на оптимізації алгоритмів розподілу партицій, запропонований підхід спрямований на керування подіями, які передують ребалансуванню, що дає змогу впливати на саму причину його виникнення

Обговорення результатів дослідження. У межах дослідження проаналізовано вплив змін складу груп споживачів на стабільність функціонування потокових систем на базі Apache Kafka. Як основні критерії оцінювання розглянуто частоту ініціації процесів ребалансування, стабільність розподілу партицій та наявність переривань у обробці повідомлень.

Проведений аналіз свідчить, що неконтрольовані зміни складу групи споживачів призводять до частих ініціацій ребалансування, що супроводжується тимчасовими зупинками обробки даних та накопиченням повідомлень у системі.у

Застосування підходу координації життєвого циклу споживачів дає змогу зменшити кількість таких подій за рахунок узгодження процесів підключення та відключення споживачів. У результаті спостерігається більш стабільний розподіл партицій і зменшення кількості переривань у обробці даних.

Отримані результати узгоджуються з сучасними дослідженнями у сфері потокових систем, однак, на відміну від існуючих підходів, запропоноване рішення спрямоване на зменшення частоти виникнення ребалансування, а не лише на оптимізацію його виконання.

Отже, за результатами виконаної роботи можна сформулювати такі наукову новизну та практичну значущість результатів дослідження.

Наукова новизна отриманих результатів дослідження – полягає у вдосконаленні підходу до підвищення стабільності потокових систем на базі Apache Kafka шляхом зменшення кількості надлишкових ребалансувань груп споживачів. Запропоновано підхід, що базується на координації життєвого циклу споживачів, який, на відміну від існуючих рішень, орієнтований не на оптимізацію алгоритмів розподілу партицій, а на зменшення частоти ініціації процесів ребалансування за рахунок узгодження операцій підключення та відключення споживачів.

Практична значущість отриманих результатів дослідження полягає у можливості застосування запропонованого підходу при проєктуванні та експлуатації потокових систем у динамічних середовищах. Використання результатів роботи дає змогу підвищити стабільність обробки даних, зменшити кількість переривань у роботі споживачів та знизити затримки в системах реального часу. Запропоновані рішення можуть бути інтегровані у мікросервісні та хмарні архітектури, де характерною є часта зміна складу сервісів.

Висновки з даного дослідження
і перспективи подальших розвідок у даному напрямі

У роботі розглянуто проблему забезпечення стабільності функціонування потокових систем у динамічних середовищах та проаналізовано особливості їх реалізації на базі Apache Kafka.

Проведено аналіз архітектури потокових систем і механізмів координації груп споживачів, що дало змогу визначити ключову роль процесу ребалансування у забезпеченні узгодженості обробки даних. Досліджено причини виникнення ребалансування та встановлено, що часті зміни складу споживачів є основним фактором, який негативно впливає на стабільність роботи системи.

Показано, що ребалансування супроводжується тимчасовими перервами в обробці повідомлень, що призводить до зростання затримок та зниження продуктивності. Особливо це проявляється у динамічних середовищах, де процеси масштабування та оновлення сервісів відбуваються регулярно.

Запропоновано підхід до зменшення кількості надлишкових ребалансувань, що базується на координації життєвого циклу споживачів. Реалізація підходу передбачає узгодження процесів підключення та відключення споживачів, що дає змогу зменшити кількість ініціацій ребалансування та підвищити стабільність розподілу партицій.

У результаті дослідження встановлено, що застосування запропонованого підходу сприяє підвищенню стабільності функціонування потокових систем, зменшенню переривань у обробці даних та покращенню ефективності використання ресурсів. Зазначимо, що запропонований підхід є ефективним у сценаріях керованої зміни складу споживачів, однак його застосування в умовах аварійних відмов потребує додаткових досліджень.

Література

1. Nasir M. A. Uddin, G. De Francisci Morales, D. Garcia-Soriano, N. Kourtellis, M. Serafini. Partial Key Grouping: Load-Balanced Partitioning of Distributed Streams // Proceedings of the VLDB Endowment. – 2015. – Режим доступу: https://arxiv.org/abs/1510.07623

2. Sharvari T., Sowmya Nag K. *A* Study on Modern Messaging Systems—Kafka, RabbitMQ and NATS Streaming [Електронний ресурс] // arXiv preprint. – 2019. – Режим доступу: https://arxiv.org/abs/1912.03715.

3. Bunrong Leang, Sokchomrern Ean, Ga-Ae Ryu, Kwan-Hee Yoo. Improvement of Kafka Streaming Using Partition and Multi-Threading in Big Data Environment // Sensors. – 2019. – DOI:10.3390/s19010134.

4. Isah H., Ughofa T. A., Mahfuz S., Ajerla D., Zulkerinne F., Khan S. A Survey of Distributed Data Stream Processing Frameworks // IEEE Access. – 2019. – DOI:10.1109/ACCESS.2019.2946884.

5. Henning S., Hasselbring W. Theodolite: Scalability Benchmarking of Distributed Stream Processing Engines in Microservice Architectures [Електронний ресурс] // arXiv preprint. – 2020. – Режим доступу: https://arxiv.org/abs/2009.00304.

6. Raptis T. P., Passarella A. On Efficiently Partitioning a Topic in Apache Kafka [Електронний ресурс] // arXiv preprint. – 2022. – Режим доступу: https://arxiv.org/abs/2205.09415

7. Raptis T. P., Passarella A. A Survey on Networked Data Streaming with Apache Kafka // IEEE Access. – 2023. – DOI:10.1109/ACCESS.2023.3303810.

8. Landau D., Andrade X., Barbosa J. G. Kafka Consumer Group Autoscaler [Електронний ресурс] // arXiv preprint. – 2022. – Режим доступу: https://www.researchgate.net/publication/361479594_Kafka_Consumer_Group_Autoscaler

9. Landau D., Saurabh N., Andrade X., Barbosa J. G. Multi-Objective Optimization of Consumer Group Autoscaling in Message Broker Systems [Електронний ресурс] // arXiv preprint. – 2024. – Режим доступу: https://arxiv.org/abs/2402.06085

10. Pfister B. J. J., Scheinert D., Geldenhuys M. K., Kao O. Daedalus: Self-Adaptive Horizontal Autoscaling for Distributed Stream Processing Systems [Електронний ресурс] // arXiv preprint. – 2024. – Режим доступу: https://arxiv.org/abs/2403.02093.

11. Landau D., Saurabh N., Andrade X., Barbosa J. G. Latency and cost-aware consumer group autoscaling in message broker systems // Journal of Parallel and Distributed Computing. – 2025. – DOI: https://doi.org/10.1016/j.jpdc.2025.105071.

12. Confluent. (2026). Consumer group protocol. Confluent Developer. Режим доступу: https://developer.confluent.io/courses/architecture/consumer-group-protocol/

References

1. Nasir, M. A. U., De Francisci Morales, G., Garcia-Soriano, D., Kourtellis, N., & Serafini, M. (2015). Partial key grouping: Load-balanced partitioning of distributed streams. Proceedings of the VLDB Endowment. Retrieved from https://arxiv.org/abs/1510.07623.

2. Sharvari, T., & Sowmya Nag, K. (2019). A study on modern messaging systems—Kafka, RabbitMQ and NATS streaming. arXiv preprint. Retrieved from https://arxiv.org/abs/1912.03715.

3. Bunrong, L., Sokchomrern, E., Ga-Ae, R., & Kwan-Hee, Y. (2019). Improvement of Kafka streaming using partition and multi-threading in big data environment. Sensors, 19(1), 134. https://doi.org/10.3390/s19010134.

4. Isah, H., Ughofa, T. A., Mahfuz, S., Ajerla, D., Zulkerinne, F., & Khan, S. (2019). A survey of distributed data stream processing frameworks. IEEE Access. https://doi.org/10.1109/ACCESS.2019.2946884.

5. Henning, S., & Hasselbring, W. (2020). Theodolite: Scalability benchmarking of distributed stream processing engines in microservice architectures. arXiv preprint. Retrieved from https://arxiv.org/abs/2009.00304.

6. Raptis, T. P., & Passarella, A. (2022). On efficiently partitioning a topic in Apache Kafka. arXiv preprint. Retrieved from https://arxiv.org/abs/2205.09415.

7. Raptis, T. P., & Passarella, A. (2023). A survey on networked data streaming with Apache Kafka. IEEE Access. https://doi.org/10.1109/ACCESS.2023.3303810.

8. Landau, D., Andrade, X., & Barbosa, J. G. (2022). Kafka consumer group autoscaler. arXiv preprint. Retrieved from https://www.researchgate.net/publication/361479594_Kafka_Consumer_Group_Autoscaler.

9. Landau, D., Saurabh, N., Andrade, X., & Barbosa, J. G. (2024). Multi-objective optimization of consumer group autoscaling in message broker systems. arXiv preprint. Retrieved from https://arxiv.org/abs/2402.06085.

10. Pfister, B. J. J., Scheinert, D., Geldenhuys, M. K., & Kao, O. (2024). Daedalus: Self-adaptive horizontal autoscaling for distributed stream processing systems. arXiv preprint. Retrieved from https://arxiv.org/abs/2403.02093.

11. Landau, D., Saurabh, N., Andrade, X., & Barbosa, J. G. (2025). Latency and cost-aware consumer group autoscaling in message broker systems. Journal of Parallel and Distributed Computing, 201, 105071. https://doi.org/10.1016/j.jpdc.2025.105071.

12. Confluent. (2026). Consumer group protocol. Confluent Developer. https://developer.confluent.io/courses/architecture/consumer-group-protocol/